JRC Research Report No. 90-18
June 1988 through May 1990

Network, System, and Status Software Enhancements for the Autonomously Managed Electrical Power System Breadboard

Protocol Specification

Grant NAG8-720

Volume 2 of 4 Volumes

Prepared by

James W. McKee
University of Alabama, Huntsville
Huntsville, Alabama 35899
(205) 895-6257

Prepared for

Norma Whitehead
EB12
NASA/MSFC
Huntsville, Alabama

Table of Contents

1. Purpose

This volume contains the specification, structured flow charts, and code listing for the protocol.

2. Introduction

The purpose of an autonomous power system on a spacecraft is to relieve humans (on the ground or in the craft) from having to continuously monitor and control the generation, storage and distribution of power in the craft. This implies that algorithms will have been developed to monitor and control the power system. The power system will contain computers on which the algorithms run. Studies, [1],[2], indicate that these computers should be physically close to the hardware they monitor and/or control. The studies also indicate that there should be one central computer system that makes the high level decisions and sends commands to and receives data from the other distributed computers. This will require a communications network and an efficient protocol by which the computers will communicate.

One of the major requirements on the protocol is that it be "real time" because of the need to control the power elements. This implies a simple protocol, short messages, and as much of the protocol implemented in dedicated hardware as possible.

The objective of this introduction is to present in a logical fashion the considerations that led to the design and development of a network protocol that is being implemented on the Autonomously Managed Power System, AMPS breadboard at NASA/MSFC. The AMPS breadboard is being used to develop and test higher level control and expert system programs being developed for power system management [3].

3. Power System Description

The power system for a spacecraft will consists of one or more of each of three functional elements. The power generation center, PGC, (e.g., solar arrays) generates the electrical power for the spacecraft. The power storage center, PSC, (e.g., banks of batteries) stores energy until needed. Load centers, LC, switch the power from the distribution busses to the loads.

An autonomous power system can be thought of as system in which the monitoring and control hardware are distributed to each functional element and are connected by a network. The power control center, PCC, is the central computer(s) on which the high level programs run. Each PGC, PSC, and LC will contain the hardware and software to monitor (and maybe control) the voltages, currents, and temperatures in the center and monitor and control the settings of the switches that connect the center to the power bus.

The AMPS test facility currently features the following.

1. A programmable solar array simulator which supplies 220 +/- 20 VDC directly to three power channels with a maximum power output of 75 kW.

2. An energy storage simulator which consists of a battery with 168 commercial nickel-cadmium (Ni-Cd) cells serially connected to provide a nominal DC voltage of 220 volts and a capacity of 189 ampere-hours.

3. A load simulator which consists of nine resistive loads and one dynamic load that dissipate a total of 24 kW of power when operated at 200 VDC.

In addition, three Motorola 68000 microcomputer based controllers provide data retrieval and low-level decision-making for the power system with a NCR Tower based host computer providing overall power system management and programmability for flight power system simulation.

4. Protocol Requirements

This section will present the list of protocol requirements that were derived from the physical layout of the power system and the monitoring and control requirements of the overall power management system. The objective in developing this protocol was to make the protocol as simple as possible and still satisfy the requirements of the power system.

The following is a list of the assumptions made that simplified the protocol.

1. The bus topology will be used. Therefore, there is no need for routing information, all stations are on the same line.

2. Once the network is initialized there will not be a need to open and close sessions, i.e., all the stations stay on the line all the time.

3. All the messages will be short, on the order of 4 to 200 bytes (our analysis, [1], and [2] support this).

4. Because the physical length of the network will be short, i.e., it will be totally confined to be within the space craft,

the protocol will not need to be as robust as some of the more common protocols.

The following is a list of the requirements of the protocol.

1. Since the power system must be controllable in real time, the protocol must be capable of processing (from the transmitting application to the receiving application) messages in times on the order of 10 ms to 100 ms. (In [2] a study was performed that indicates for a power system to have a reaction time of 0.1 seconds a communication bit rate of on the order of 10^6 is needed.)

2. The protocol must be able to initialize or re-initialize itself.

3. The protocol must be able to add or remove stations at will.

4. Since the functional elements are physically dispersed, the protocol must have the capacity to uniquely address an arbitrary number of stations.

5. Since this is a control system the protocol must insure that each message is delivered to the application in the order sent.

Given these requirements the question arises "Is it possible to use a defined protocol such as TCP/IP [4] or DDCMP [5]?" There are two disadvantages to using TCP/IP: 1) the amount of computer time needed to process each message and 2) the number of over head bytes in each message (minimum of 40). The major draw back of TCP/IP is, of course, the amount of computer time required. For this reason TCP/IP was rejected. Since the power system environment is more constrained then the environment for which DDCMP was designed, the protocol did not need all the capabilities of the message exchange section of DDCMP.

The protocol developed is on the level of DDCMP. Although not a requirement, the protocol assumes that it is working on top of Ethernet hardware such as the ENP-30 card [6],[7]. This card performs two of the three functions of DDCMP: framing and link management (as does most network hardware). Some of but not all of the message exchange features were incorporated. The ability to use logical station addresses was added.

5. Protocol Description

This section will give an over view of the protocol. For a detailed description of the protocol see the specification in Appendix A. The following is a list of the main attributes of the protocol.

1. The PCC initializes the network and establishes a session with each other element on the power system network.

2. The network is self-initializing and can re-synchronize itself.

3. Every packet is numbered.

4. Messages will be passed to the application program in the correct order.

5. Every message is acknowledged.

6. There is a sliding acknowledgement window.

7. Messages are capable of being pipelined, i.e., messages can be accepted before previous acknowledgements reach the sender.

8. There is a re-transmission of messages that have not been acknowledged within the time out period.

9. There is a mapping between logical power system elements' names and physical Ethernet addresses.

The computer system in each power system element is doing two distinct operations: processing communication messages and performing its specific monitor and/or control functions. The present breadboard uses a simple scheme of two CPUs running in parallel. One runs the protocol; the other runs the application program. The two computers communicate through shared memory. This is faster and simpler than one CPU with a complex operating system that supports multi-processing.

The protocol is designed to be the interface between application programs and the Ethernet hardware. The application programs generate messages that are to be sent to other stations. They pass these messages to the protocol. The Ethernet hardware does the actual transmitting and receiving of messages over the physical wire. To transmit a message the Ethernet hardware is given the packet to be sent. The hardware sends the packet over the cable and is responsible for assuring that the message is transmitted correctly, i.e., it will retransmit the packet if it detects a collision. When the Ethernet hardware receives a packet, it checks the unique station address and cyclic redundancy code, CRC, and only accepts packets which are addressed to the station and in which no errors have been detected.

The philosophy of the implementation of the protocol is: messages are never moved around; only pointers to the messages are moved. This resulted in an implementation based on stacks and queues and results in the ability of the protocol to quickly process messages.

On a very high level the protocol can be viewed as follows. For the receive function: the Ethernet hardware stores a received message in the shared memory of both processors and passes a pointer to the message to the protocol; the protocol passes the pointer to the message to the application program. For the transmit function: the application program creates a message in shared memory and passes a pointer to the message to the protocol which in turn passes the pointer to the Ethernet hardware which transmits the message.

The protocol first initializes the network then moves round robin between three states: transmit state, receive state, and time out state. The protocol in the PCC initializes the network by sending a packet, in broadcast mode, requesting all other stations to identify themselves. Each station sends back its logical name and Ethernet address. Then in each station the protocol will enter the transmit state and check if there is a message to send. If so the message will be sent. If not the protocol will enter the receive state and check if there are messages to be moved to the application program. If so, the messages are moved. If not, the protocol will enter the time out state. If a time out has occurred, unacknowledged messages are sent again and the timer is reset. Then the protocol enters the transmit state, etc.

When the application has a message ready to be transmitted, it sets the Go-Flag. When the protocol enters the transmit state, the protocol tests the Go-Flag and when set makes a packet out of the message. The protocol resets the Go-Flag which indicates to the application that the message has been sent. The protocol combines the pointer to the message with a unique packet number and other data needed by the Ethernet hardware to make a packet. The protocol then passes a pointer to the packet to the Ethernet hardware. The hardware transmits the packet (but does not remove the packet from memory) and places the pointer to the packet on the wait for acknowledgement stack, WFASk.

As packets are received by the Ethernet hardware they are placed in shared memory and a pointer to the packet is placed on the receive control block stack, RCBSk. The protocol monitors the RCBSk and processes any packets found in the queue. The packets can be either an acknowledgement or a message. If the packet is an acknowledgement, the WFASk is checked for corresponding message packet(s). (There is a sliding acknowledgement window on packet numbers, so more than one packet can be acknowledged with only one acknowledgement packet.) All message packets in the WFASk queue that have been acknowledged are deleted from the WFASk and memory. If the packet is a message packet, the unique packet number is checked against the expected number for the packet. If the numbers are the same the packet is broken apart and the pointer to the message is placed on the command buffer stack, CBSk. If the numbers are not the same, the pointer is placed back at the end of the RCBSk queue. (A packet has been received before its predecessor has been correctly received.) Every message packet is acknowledged. The protocol creates a acknowledgement packet which is transmitted by the hardware. (Acknowledgement packets are automatically removed from the WFASk.) The application program monitors the CBSk. When it detects messages in the CBSk queue, it processes the messages in the order in which the messages are on the CBSk. (This insures that messages are processed in the order sent by the other station.)

The time-out function re-transmits any packets that are on the WFASk when a time out occurs. If the packet is not acknowledged within the time out period, the packet's pointer is taken off the WFASk and passed to the hardware to re-transmit the packet. The Ethernet hardware puts the pointer back on the WFASk. When the packet is acknowledged, its pointer is deleted from the WFASk and the message memory freed.

6. References

[1] TRW, "Space Power Distribution System Technology, Final Report," Vol. 2, 1983, TRW Report No. 34579-6001-UT-00.

[2] Martin Marietta Aerospace, "Space Station Automation of Common Mode Power Management and Distribution, Interim Final Report," 1989, MCR-89-516.

[3] Weeks, D.J., "Expert Systems in Space," IEEE Potentials, Vol. 6, No. 2, 1987.

[4] Tanenbaum, A. S., Computer Networks, Prentice Hall, 1988.

[5] DEC, "Digital Data Communications Message Protocol, DDCMP," (Specification), March 1, 1978, AA-D599A-TC.

[6] Communication Machinery Corporation, "Ethernet Node Processor, ENP-30 Users Guide," 1985.

[7] Communication Machinery Corporation, "Ethernet Node Processor, K-1 Kernel Software User's Guide," 1985.

7. Protocol Definitions

Buffer is an array of storage in which node address, status, command names, and data are stored.

Control block is an array of storage which contains control information to/from the Kernel and an address of a buffer.

Command is a message to level 7 of the network. In this system a command is generated and interpreted by the system software.

Queue is a type of stack in which the bottom item is the next item accessed, i.e., a circular stack in which items are put in one end and taken out the other.

Function is the software that generates and interpret commands.

Node is a sender/receiver on the network.

LIFO is a type of stack in which the top item is the next item accessed, i.e., a push down stack.

Packet is a message to level 3 of the network. In this system a packet is interpreted by the protocol software. Packets usually contain commands. But there are packets that are used only by the protocol software and never seen by the system software, e.g., an acknowledgement packet. A packet consists of a control block and a buffer.

Stack is an array of storage with associated pointer to indicate the start of the stack, the end of the stack, and where to access data in the stack.

Station consists of a node and the computer and other hardware that interface and control the power hardware.

The following is a list of the abbreviations and their definitions.

general

NM, network manager
LCC, load center controller
PSC, power source controller
RPC, remote power controller
EPSC, electrical power system controller

addresses

DNAd, destination node address
RBAd, receive buffer address
SBAd, send buffer address
SNAd, sending node address

arrays -- these arrays store the present status values for the system

BVAr, battery voltage array
LCDTAr, LC diode temperature array
LCPAr, LC power array
PSPAr, PS power array
PSTAr, PS temperature array
SDAr, switch data array

array pointers -- points to the start of the corresponding array

BVAPt, battery voltage array pointer
LCDTAPt, LC diode temperature array pointer
LCPAPt, LC power array pointer
PSPAPt, PS power array pointer
PSTAPt, PS temperature array pointer
SDAPt, switch data array pointer

blocks --

ICBk, initialization command block

IRBk, initialization response block

RCBk, receive control block
A RCBk is used by the Kernel to pass the information about a received packet. The RCBk contains the address of the buffer in which the Kernel placed the data of the packet. There will be RCBCn (receive control block constant) number of RCBks that physically reside in the RAM on the ENP-30 card. Table 4 shows the definition of the fields in a RCBk.

SCBk, status control block

TCBk, transmit control block
A TCBk contains the information needed for the Kernel to form and transmit a packet. A TCBk contains the address of the buffer containing the data to be transmitted. There will be TCBCn (transmit control block constant) number of

TCBks that physically reside in the RAM on the ENP-30 card. The location of the fields in the TCBk is shown in Table 3.

TOCBk, time out control block
A TOCBk contains the information needed for the timer portion of the Kernel. The location of the fields in the TOCBk are shown in Table 5.

constants

ALCn	address length constant -- number of bytes in an address (2)
ANACn,	active node address constant -- size of ANASk (16)
BCn,	buffer constant -- number of buffers (32)
BLCn,	buffer length constant -- number of bytes in a buffers (256)
CBSCn,	command buffer stack constant -- number of addresses locations in the command buffer stack (4)
HLCn,	header length constant -- number of bytes in the header (16)
ITCBSCn,	idle transmit control block stack constant -- number of address locations on the idle transmit control block stack (20)
RCBCn,	receive control block constant -- number of RCBks (16)
RCBSCn,	receive control block stack constant -- number of address locations on the receive control block stack (20)
SNCn,	station name -- the unique name of the station, i.e., LCC1, EPSC, etc.
TCBCn,	transmit control block constant -- number of TCBks (16)
TOCn,	timeout constant -- number of 2 ms. increments of time between timeouts (2)
WFASCn,	wait for acknowledgement stack constant -- number of address locations in the wait for acknowledgement stack (20)

fields

AkFd, acknowledgement field
ANAFd, active node address field
ANNFd, active node name field
ANRPNFd, active node receive packet field
ANTPNFd, active node transmit packet field
BAFd, buffer address field
CNFd, command name field
DAFd, destination address field
ICBESAFd, initialization command block Ethernet station address field
ICBLAFFd, initialization command block logical address filter field
ICBMFd, initialization command block mode field

ICBNRDFd, initialization command block number receive descriptor field
ICBNTDFd, initialization command block number transmit descriptor field
ICBRIHAFd, initialization command block receive interrupt handler address field
ICBTIHAFd, initialization command block transmit interrupt handler address field
IRBESAFd, Initialization response block Ethernet station address field
IRBSRAFd, Initialization response block status routine address field
IRBRRAFd, Initialization response block receive routine address field
IRBTRAFd, Initialization response block transmit routine address field
IRBTORAFd, Initialization response block timer routine address field
DLFd, data length field
PNFd, packet number field
RAFd, receive address field
RBAFd, receive buffer address field
RBLFd, receive buffer length field
RBSFd, receive buffer status field
SFd, select field
SAFd, source address field
SCBFCFd, status control block function code field
SCBRFd, status control block return field
SCBSBAFd, status control block statistics block address field
SNFd, station name field
TBAFd, transmit buffer address field
TBLFd, transmit buffer length field
TSFd, time stamp field
TOFd, timeout field
TOECFd, timeout event count field
TOSAFd, timeout subroutine address field

<u>flags</u>

PRFg, protocol ready flag
RTFg, retransmit flag
SBFg, send buffer flag

<u>Numbers</u> -- constants

ANANo, active node address number
CNo, command number
RBNo, receive buffer numbers
RPNo, receive packet number
TBNo, transmit buffer number

TPNo, transmit packet number

Offsets

ANOs,	active node offset (9 bytes)
ANAFOs,	active node address field offset (0 bytes)
ANNFOs,	active node name field offset (6 bytes)
ANTPNFOs,	active node transmit packet number field offset (7 bytes)
ANRPNFOs,	active node receive packet number field offset (8 bytes)
DAFOs,	destination address field offset (0 bytes)
SAFOs,	source address field offset (6 bytes)
AkFOs,	acknowledgement field offset (12 bytes)
SNFOs,	station name field offset (13 bytes)
PNFOs,	packet number field offset (14 bytes)
DLFOs,	data length field offset (16 bytes)
CNFOs,	command name field offset (18 bytes)
SFOs,	select field offset (20 bytes)
DOs,	data offset (22 bytes)
RBAFOs,	receive buffer address field offset (8 bytes)
RBLFOs,	receive buffer length field offset (6 bytes)
RBSFOs,	receive buffer status field offset (4 bytes)
RMLFOs,	receive message length field offset (12 bytes)
RTFOs,	retransmit flag offset (6 bytes)
TBAFOs,	transmit buffer address field offset (8 bytes)
TBLFOs,	transmit buffer length field offset (6 bytes)
TOSAFOs,	time out subroutine address field offset (8 bytes)
TOECFOs,	time out event code field offset (12 bytes)

stacks and queues

ANASk, active nodes address stack -- LIFO (ANACn * ANOs bytes)

The ANASk will contain information on the nodes that are communicating with this node. The ANASk will contain four fields for each active node: active node address field, ANAFd, active node name field, ANNFd, active node transmit packet number field, ANTPNFd, and active node receive packet number field, ANRPNFd. This list will be ANACn nodes deep. The ANAFd contains the 6 byte address of a node to which this node is communicating. The ANNFd contains the unique name (number) of the active node, see table 2. The ANTPNFd will contain the transmit packet number, TPNo, for the number of the next packet to be transmitted. The ANRPNFd will contain the receive packet number, RPNo, for the number of the next packet to be received from the address. [The TPNo will be inserted into the buffer before the packet is transmitted. The RPNo will be compared to each packet from the address. If the packet number is not the same as the RPNo, the command will be ignored. The packet will be acknowledged.]

	ANAFd (6 bytes)	ANNFd (1 byte)	ANTPNFd (1 byte)	ANRPNFd (1 byte)
active node	6 byte address	unique name	variable	variable
active node	6 byte address	unique name	variable	variable
o o o				
active node	6 byte address	unique name	variable	variable

CBSk, command buffer stack -- queue (ALCn * CBSCn bytes)
The CBSk contains the addresses of the buffers which contain the commands that are waiting to be processed. The commands are processed in a first in - first out fashion. The CBSOt points to the next buffer to be processed. The CBSIn points to the where the next buffer address will be stored.

ITCBSk, idle transmit control block stack -- LIFO (ALCn * ITCBSCn bytes)
The ITCBSk contains the addresses of TCBks that are not in use. When a TCBk is needed it is popped off this stack.

IBSk, idle buffer stack -- LIFO (ALCn * BCn bytes)
The IBSk contains the addresses of the buffers not in use. There will be BCn (buffer constant) of buffers. Each buffer will be BLCn (buffer length constant) bytes long. The buffers will reside in the ENP-30's RAM.

RCBSk, receive control block stack -- queue (RCBSCn * ALCn bytes)
The RCBSk contains the addresses of the RCBks of received packets.

WFASk, waiting for acknowledgement stack -- queue (WFASCn * ALCn bytes)
The WFASk contains the address of TCBks of packets that have not been acknowledged. When a packet is acknowledged the corresponding TCBk is replaced with the last TCBk (pointed to by WFASOt) on the stack. The WFASOt is incremented.

<u>stack pointers</u>

head --- points to the start of a stack, the smallest absolute address, never changes
tail --- points to the end or top of a stack, the largest absolute address, never changes

in --- points to the location in which to store the next entry in a queue, increases up to tail then reset to head
out --- points to the location from which to get the next piece of data in a queue, increases up to tail then reset to head
push/pop- points to the location for the top of the LIFO stack, pop from it, push at 1 + top
modules increment -- increment pointer, if greater than tail set equal to head

ANASHd, active nodes address stack head
ANASPp, active nodes address stack push/pop
ANASTl, active nodes address stack tail
CBSHd, command buffer stack head
CBSIn, command buffer stack in
CBSOt, command buffer stack out
CBSTl, command buffer stack tail
ITCBSHd, idle transmit control block stack head
ITCBSPp, idle transmit control block stack push/pop
ITCBSTl, idle transmit control block stack tail
IBSHd, idle buffer stack head
IBSPp, idle buffer stack push/pop
IBSTl, idle buffer stack tail
RCBSHd, receive control block stack head
RCBSIn, receive control block stack in
RCBSOt, receive control block stack out
RCBSTl, receive control block stack tail
WFASHd, waiting for acknowledgement stack head
WFASIn, waiting for acknowledgement stack in
WFASOt, waiting for acknowledgement stack out
WFASTl, waiting for acknowledgement stack tail

subroutine calls to the Kernel

KINIT, call to the Kernel initialize routine. pass address of ICBk
KOUT, call to the Kernel timeout routine. pass address of TOCBk
KRCV, call to the Kernel receive routine. pass address of RCBk
KSTS, call to the Kernel control/status routine. pass address of SCBk
KXMT, call to the Kernel transmit routine. pass address of TCBk

8. Protocol design specification

8.1. Level 1 and part of level 2 of the protocol will be implemented by the ENP-30 board or equivalent.

8.2. Node to node protocol -- Each packet will contain the 48 bit destination node address, DNAd, contained in the destination address field, DAFd, for the packet and the 48 bit source node address, SNAd, contained in SAFd of the packet, as shown in table 1. A node will only process packets addressed to it.

8.3. Each packet will have a packet number field, PNFd. There will also be an acknowledgement field, AkFd. The node that originated the command will place in the PNFd the packet's number and set the AkFd to indicate that the packet is a command. The receiving node will return to the sender a packet that contains in the PNFd the number of the packet sent and the AkFd set to indicate an acknowledgement. If a packet is not acknowledged within the time-out interval, the packet will be sent again.

8.4. The protocol will use the following instructions to establish a network.

Reset Network -- the receipt of this command will cause the node to reset itself, in particular the node will do the following:

A. Clear its ANASk (move the head pointer to the start of the stack);
B. The return of an acknowledgement packet is optional;
C. Clear the WFASk;
D. Clear the RCBSk;
E. Clear the CBSk.

Network initialize [w/o-ack] (with out acknowledgement)-- will initialize the network. It will be sent by the network manager. Each receiving node will do the following:

A. Return a network initialize [ack] packet;
B. The return of an acknowledgement packet is optional;
D. Place the data from the sending node on the ANASk;

Network initialize [w-ack] (with acknowledgement) -- will place the data from the sending node into the ANASk of the receiving node. An acknowledgement packet is required.

8.5. Network manager -- The EPSC will be designated as the network manager. Whenever the network is initialize, the network manager, NM, will send out in, broadcast mode, a network initialize [w/o-ack] packet. This packet requests that each node on the network send the NM the receiving node's name and address. Each node must have embedded in its software a unique name, e.g., LCC1, LCC2, PSC1, PSC2, expert system, etc. This is necessary to enable the EPSC to control the individual stations of the system if there are more than one of each type of station on the network. When a node receives a network initialize [w/o-ack] packet, the node will send the node's address and name in a initialize network [w-ack] packet until the packet is acknowledged. After the network is initialized, each time a node receives a packet, it will compare the contents of the SAFd to the contents of each ANAFd in the ANASk. If there is not a match, the packet is ignored.

8.6. The description of the protocol will be divided into five sections: general information, initialization, transmit state, receive state, and timeout state. The protocol is based on the philosophy of a stack of buffers in which data is stored, and the moving of the addresses of these buffer. Once data is received or generated in the node, the data is not copied to any other buffer; the pointer to the buffer is moved. There is a set of stacks between which the addresses of the buffers are moved. There are also flags which are used to indicate the status of portions of the protocol.

General information

After initialization the protocol is in a loop between three states: transmit state, receive state, and timeout state. The protocol loops through the testing of the SBFg, testing of the RTFg, and entering the receive state.

The SBFg is set to one by functions in the operating system when a buffer is ready to be sent to another station. The buffer will contain all the information necessary to form a packet. If the SBFg is set, then the protocol will enter the transmit state and form and transmit a packet of data to the desired node.

The protocol always enters the receive state. It then cleans up the WFASk and processes any packets on the RCBSk. When a node receives a packet, the Kernel places the RCBk on the RCBSk. The packet could be either an acknowledgement, a command, or one of the network initialize packets.

The Kernel has an internal clock that will, by setting the RTFg to one, inform the protocol when it is necessary to resend packets that have not been acknowledged. If the RTFg is set, the protocol will retransmit any packets that have not been acknowledged.

Initialization

On power up or reset, the Kernel initializes itself and the Lance. Then the Kernel waits for the operating system to down load the protocol software and set the go bit in the Kernel's mailbox. Once the go bit has been set, the Kernel passes control of the ENP-30 microprocessor to the protocol software. The following is a list of the operations necessary to initialize the protocol:

A. The protocol calls the Kernel's initialization command. This command returns the addresses of the Kernel's status subroutine, receive subroutine, transmit subroutine, timer subroutine, and Ethernet node address.

B. All the stack pointers are set to the start of their respective stacks.

C. The addresses of all the buffers are placed on the IBSk.

D. For each RCBk a buffer address is popped off IBSk and placed in the RBAFd. Each RCBk is passed to the Kernel through a receive subroutine call.

E. The ANASk is cleared.

F. The RTFg and SBFg are set to zero.

G. All the TCBks are cleared and placed in the ITCBSk.

H. The Kernel's status subroutine is called, which starts the Lance. This enables the node to start to receive and transmit packets.

I. The PRFg is set to one. This enables the operating system to continue.

Only in the EPSC will the following be implemented.

J. The EPSC initializes the network by making and sending a network initialize [w/o-ack] packet as follows:
   a. A TCBk is popped off the ITCBSk;
   b. A buffer is popped off the IBSk and the address placed in the TBAFd of the TCBk.
   c. The contents of HLCn is placed in the TBLFd.
   d. The following data is placed in the fields of the buffer:
      1. The broadcast Ethernet address (all 1's) is placed in DAFd;
      2. The node's Ethernet address will be placed in the SAFd;
      3. AkFd will be set to indicate a network initialize [w/o-ack] packet;
      4. The station name (number) (see table 2) will be placed in the SNFd;
      5. The PNFd is set to zero.
   e. Then the address of the TCBk is passed to the Kernel through a transmit subroutine call.

Transmit State

If the SBFg is a one when checked, the protocol will enter the transmit state. The data flow diagram for the transmit state is shown in figure 1. The protocol uses a TCBk to make a packet. The last two TCBks on the ITCBSk are reserved for use by the receive state protocol.

A. Therefore, if ITCBSPp - ITCBSHd is less than three, a packet can not be made, and the protocol exits the transmit state.

B. If more than two TCBks are on the ITCBSk, then the protocol makes a packet as follows:
   a. A TCBk is popped off the ITCBSk;
   b. The buffer address in SBAd is transferred to the TBAFd of the TCBk. (See table 3 for a description of the fields in the TCBk.)
   c. The length of the data in the buffer (DLFd + HLCn) is placed in the TBLFd.
   d. The following data is placed in the respective fields of the buffer if the contents of the LDFd match the contents of an ANNFd:
      1. DAFd will be set to the contents of the ANAFd;
      2. The node's Ethernet address will be placed in the SAFd;
      3. AkFd will be set to indicate a command;
      4. The station name (number) (see table 2) will be placed in the SNFd;
      5. The TPNo from the ANTPNFd of the ANASk for the receiving node (content of the DAFd equal content of ANAFd) will be placed in the PNFd.
      6. And TPNo will be modules incremented.
   e. Then the address of the TCBk is passed to the Kernel through a transmit subroutine call.
   f. The SBFg is set to zero.

C. The protocol exits the transmit state.

After the packet has been transmitted, the Kernel places the address of the TCBk on the WFASk.

Clean up State

The protocol will in a round-robin fashion enter the clean upstate. In the clean up state the protocol will clean up the WFASk.

The protocol removes all the acknowledgement packets or network initialize [w/o-ack] packets from the WFASk. Starting at the TCBk pointed to by WFASOt, the AkFd of the buffer of each TCBk is examined. If it is an acknowledgement or a network initialize [w/o-ack] packet, the packet is broken apart:

- a. The TCBk is pushed onto the ITCBSk;
- b. The contents of the TBAFd is pushed onto the IBSk;
- c. The location of the TCBk in the WFASk is filled with the TCBk pointed to by WFASOt.
- d. And WFASOt is modules incremented.

The protocol then exits the clean up state.

Receive State

The protocol will in a round-robin fashion enter the receive state. In the receive state the protocol will process any commands on the RCBSk and update, if necessary, the ANASk. The data flow diagram for the receive state is shown in figure 2.

The Kernel maintains a stack of addresses of idle RCBks. When the Lance receives a packet the Kernel will supply a RCBk to Lance. Lance places the data in the buffer of the RCBk. The Kernel will then, through an interrupt, place the address of the RCBk on RCBSk.

A. The protocol starts processing the RCBks on the RCBSk. The protocol starts at the RCBk pointed to by RCBSOt and processes each RCBk up to RCBSIn. First each packet is broken apart.

- a. If bit 15 of the RBSFd is a zero, then there has not been an error in the reception of the packet in the Lance and the packet can be used. The following is performed.
  1. The buffer address in RBAFd is placed in RBAd.
  2. A buffer address is popped off of IBSk and placed in RBAFd.
- b. The RCBk is passed to the Kernel in a receive subroutine call.

Now the AkFd of the buffer in RBAd is examined. The buffer can contain either an acknowledgement, a network initialization, or a command.

B. If the buffer of RBAd is an acknowledgement, the protocol searches the WFASk looking for a corresponding command. (The search is between the TCBk pointed to by WFASOt up to the TCBk pointed to by WFASIn.)

- a. For each command on the WFASk, (the TCBks can be for either command or acknowledgement packets) the protocol finds the transmitting node on the ANASk as follows:

If the content of the SAFd of the packet equals the content of the DAFd of the buffer in RBAd, then the protocol checks the RPNo as follows:

If the content of the PNFd of the packet is at least as large but not greater than four more than the content of the PNFd of the buffer, the TCBk has been acknowledged. (A sliding acknowledgement window of four.)

a) the TCBk is pushed onto the ITCBSk;
b) The buffer address in TBAFd is pushed onto the IBSk;
c) The location of the TCBk in the WFASk is filled with the TCBk pointed to by WFASOt;
d) And WFASOt is modules incremented.

b. The address in RBAd is pushed onto the IBSk.

C. If the buffer is a network initialize command, the protocol checks if the transmitting node is on the ANASk as follows:

a. If the content of the SAFd of the buffer is not equal to the content of any of the ANAFds, then the protocol adds the new node to the ANASk as follows:
   1. The content of the SAFd is moved to ANAFd;
   2. The content of the SNFd is moved to the ANNFd;
   3. The content of the PNFd is moved to the ANRPNFd;
   4. The ANTPNFd is set to zero.
b. If the buffer is a network initialize [w/o-ack], then the protocol creates an network initialize [w-ack] as follows:
   1. Pop a TCBk off the ITCBSk.
   2. Pop a buffer address off the IBSk and place the address in the TBAFd of the TCBk.
   3. Set the TBLFd to HLCn;
   4. Set the fields of the buffer as follows;
      A) Move the content of the SAFd of the RBAd buffer to the DAFd of the buffer of the TCBk.
      B) Set the AkFd of the TCBk to network initialize [w-ack].
      C) Place the Ethernet address of the node in the SAFd.
      D) Place the station name (number) (see table 2) in the SNFd.
   5. Then the address of the TCBk is passed to the Kernel through a transmit subroutine call.
c. The address in RBAd is pushed onto the IBSk.

D. If the buffer is a command, the CBSk is checked to see if it is full. If CBSk is full the command is ignored. The buffer address in RBAd is pushed onto IBSk.

(An acknowledgement packet is created and sent to the transmitting node as follows:

a. Pop a TCBk off the ITCBSk.
b. Pop a buffer address off the IBSk and place the address in the TBAFd of the TCBk.
c. Set the TBLFd to HLCn.
d. Set the fields of the TCBk buffer as follows:
   1) Move the content of the SAFd of the RBAd buffer to the DAFd of the buffer of the TCBk.
   2) Move the content of the PNFd of the RBAd buffer to the PNFd of the buffer of the TCBk.
   3) Set the AkFd of the TCBk to acknowledge.
   4) Place the Ethernet address of the node in the SAFd.
   5) Place the station name (number) (see table 2) in the SNFd.
e. Then the address of the TCBk is passed to the Kernel through a transmit subroutine call.

)

F. If the buffer is a command and the CBSk is not full, the command is checked to determine if it should be placed on the CBSk as follows:
   a. If the content of the SAFd of the RBAd buffer is equal to the content of a ANAFd, then the RPNo is checked as follow:
      1. If the content of the PNFd of the RBAd buffer is equal to the content of the ANRPNFd, then the buffer contains the expected command.
         A) The buffer address in RBAd is pushed onto CBSk at CBSIn;
         B) CBSIn is modules incremented;
         C) And the RPNo of the ANRPNFd is modules incremented.
         D) An acknowledgement packet is generated (see Acknowledgement above).
      2. If the content of the PNFd of the RBAd buffer is less but within four (a sliding window of four) of the content of the ANRPNFd, this is an old packet that has already been processed and must be acknowledged.
         A) An acknowledgement packet is generated (see Acknowledgement above).
      3. Otherwise the address in RBAd is pushed onto the IBSk.
   b. Otherwise the address in RBAd is pushed onto the IBSk.

G. After the protocol goes through this processing, it exits the receive state.

Timeout State

The Kernel has a user setable timer that counts down to zero. The user sets the initial count in the timer. When the count reaches zero, the Kernel will set a flag and call a subroutine in the user's code. The timer has a resolution of 2 milliseconds. Figure 3 shows the data flow diagram for the time out portion of the protocol.

When the RTFg is checked and is one, the protocol will call the function that will reset the RTFg, start the time out clock again, and retransmit all the packets, if any, in the WFASk.

A. The RTFg is set to zero.

B. There will be only one TOCBk. Table 5 shows the fields in the TOCBk. The timer subroutine in the Kernel is called and passed the address of the TOCBk; nothing is changed in the TOCBk except that the RTFg is reset to zero. This starts the Kernel counting on the next time out interval. The TOFd of the TOCBk contains the number of 2 ms. increments of time to be counted down. When the count reaches zero, bit 15 of the RTFg in the TOCBk is set to 1.

C. If there are any TCBks on the WFASk, they are retransmitted. The position of the WFASIn is noted, and all TCBks between WFASOt and the old WFASIn are sent to the Kernel one at a time through transmit subroutine calls. (The old WFASIn must be noted because after each TCBk is sent to the Kernel, the Kernel will place the TCBk back on the WFASk.) (The call to a subroutine in the user's program by the Kernel is not used; only a subroutine return is coded.)

D. The protocol then exits the timeout state.

Table 1 Packet Format

Hex Address	Field Name		Number of Bytes
0	Destination address	DAFd	6 bytes
6	Source address field	SAFd	6 bytes
C	Acknowledgement field	AkFd	1 bytes
D	Packet number field	PNFd	1 bytes
E	Station name field	SNFd	1 bytes
F	Logical Destination	LDFd	1 bytes
10	Forward Address field	FAFd	1 bytes
11	Return Address field	RAFd	1 bytes
12	Time stamp field	TSFd	4 bytes
16	Data length field	DLFd	1 bytes
17	Command name field	CNFd	1 bytes
18	Select field	SFd	2 bytes
1A	Data		variable

AkFd, Acknowledgement field contains a number indicating the type of packet as follows:

type		AkFd
acknowledgement	AkCn	0
command	CCn	1
network initialize [w/o-ack]	NICn	2
network initialize [w-ack]	NIACn	3
network clear	NICl	4

PNFd, The packet number field is the number of the packet.

SNFd, The station name field contains the name of the station sending the packet. Each stations has a unique number corresponding to the name of the station defined as follows:

Table 2 Station names

station type	station number 1	2	3	4	5	6	7	8
EPSC	0							
PS	8	9	10	11	12	13	14	15
LCC	16	17	18	19	20	21	22	23
Expert Sys.	24							

TSFd, The time stamp field contains the relative system time when the command is performed. The time is the number of ticks on the system clock. The system clock has a resolution of 2 ms.

CNFd, The command name field contains the name of the command.

DLFd, The data length field contains the number of bytes of data in the command and must be less than 240 (BLCn - HLCn) bytes.

Table 3 Transmit Control Block

byte address	name of block	present contents
0	Link address (address of the next	supplied by user
2	block for a multi-block packet)	
4	Status	supplied by ENP
6	Transmit buffer length field TBLFd	supplied by user
8	Transmit buffer address field	supplied by user
A	TBAFd	
C	TDR value (data used if error)	supplied by ENP
E	Reserved	used by ENP

Table 4 Receive Control Block

byte address	name of block	contents
0	Link address (address of the next	supplied by ENP
2	block for a multi-block packet)	
4	Status RBSFd	supplied by ENP
6	Buffer length RBLFd	supplied by user
8	Receive buffer address field	supplied by user
A	RBAFd	
C	Receive message length field RBLFd	supplied by ENP
E	Reserved	used by ENP

Table 5 Time Out Control Block

byte address	name of block	contents
0	Link address (address of the next	supplied by ENP
2	block for a multi-block chain)	
4	Retransmit flag, RTFg	set by ENP
6	Time Out Field TOFd	TOCn
8	Time Out Subroutine Address Field	supplied by user
A	TOSAFd	
C	Time Out Event Code Field TOECFd	supplied by user
E	Reserved	used by user

Table 6 Initialization Command Block

byte address	name of block		contents
0	Mode	ICBMFd	supplied by user
2	# Receive Descriptors	ICBNRDFd	supplied by user
4	# Transmit Descriptors	ICBNTDFd	supplied by user
6	Reserved		
8	Logical Address Filter	ICBLAFFd	supplied by user
A			
C			
E	Receive Interrupt Handler		supplied by user
10	Address	ICBRIHAFd	
12	Transmit Interrupt Handler		supplied by user
14	Address	ICBTIHAFd	
16	Bus Interrupt Handler		supplied by user
18	Address	ICBBIHAFd	
1A	Ethernet Station Address	ICBESAFd	supplied by user
1C			
1E			

Table 7 Initialization Response Block

byte address	name of block		contents
0	Ethernet Station Address	IRBESAFd	supplied by ENP
2			
4			
6	Reserved		
8	Status Routine Address	IRBSRAFd	supplied by ENP
C	Receive Routine Address	IRBRRAFd	supplied by ENP
10	Transmit Routine Address	IRBTRAFd	supplied by ENP
14	Timer Routine Address	IRBTORAFd	supplied by ENP
18	Address	ICBBIHAFd	supplied by ENP

Table 8 Status Control Block

byte address	name of block		contents
0	Function Code	SCBFCFd	supplied by user
2	CSR0 Return	SCBRFd	supplied by ENP
4	Statistics Block Address	SCBSBAFd	supplied by ENP
6			

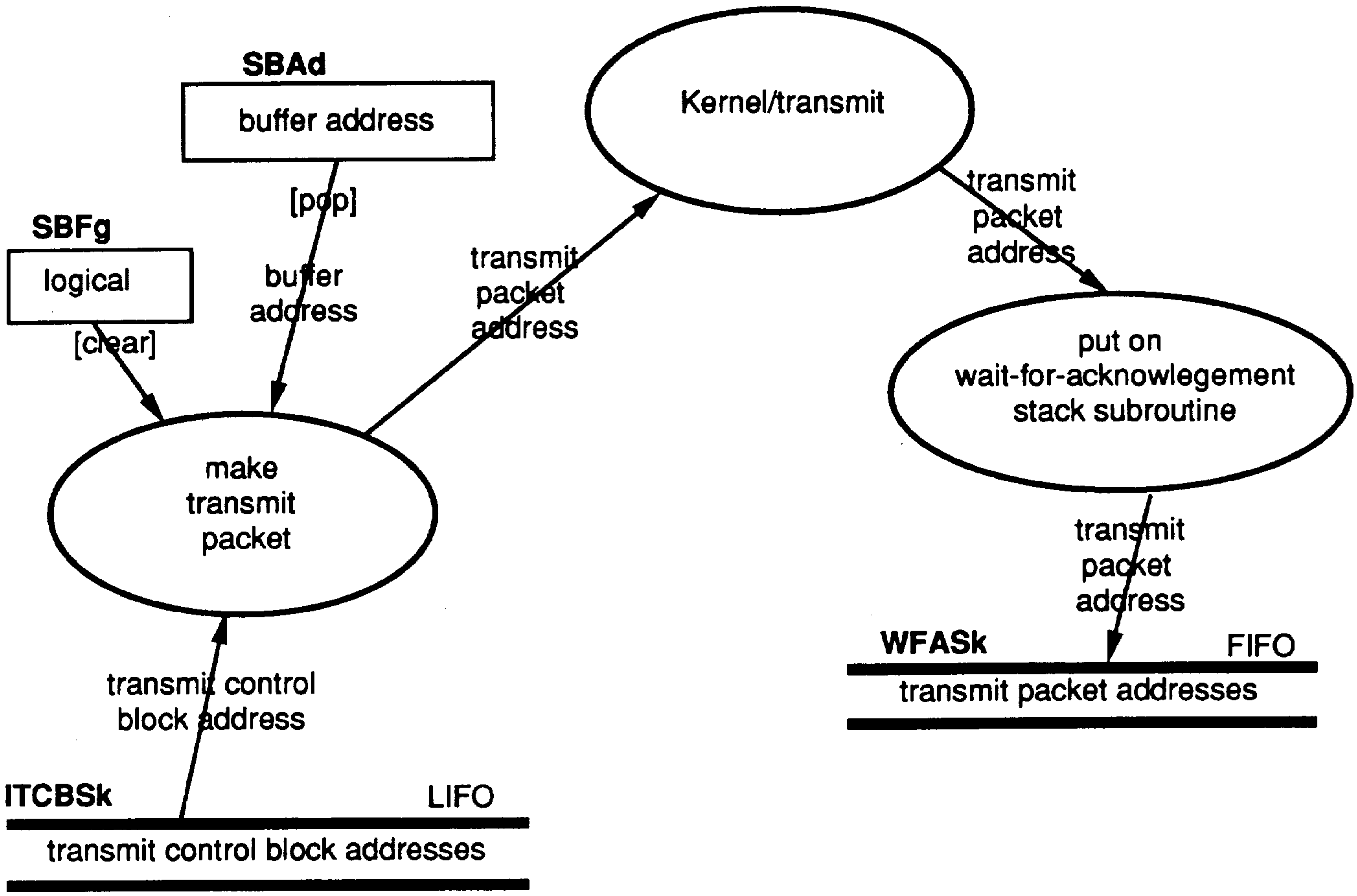


Figure 1 Data Flow Diagram-- Transmit Protocol

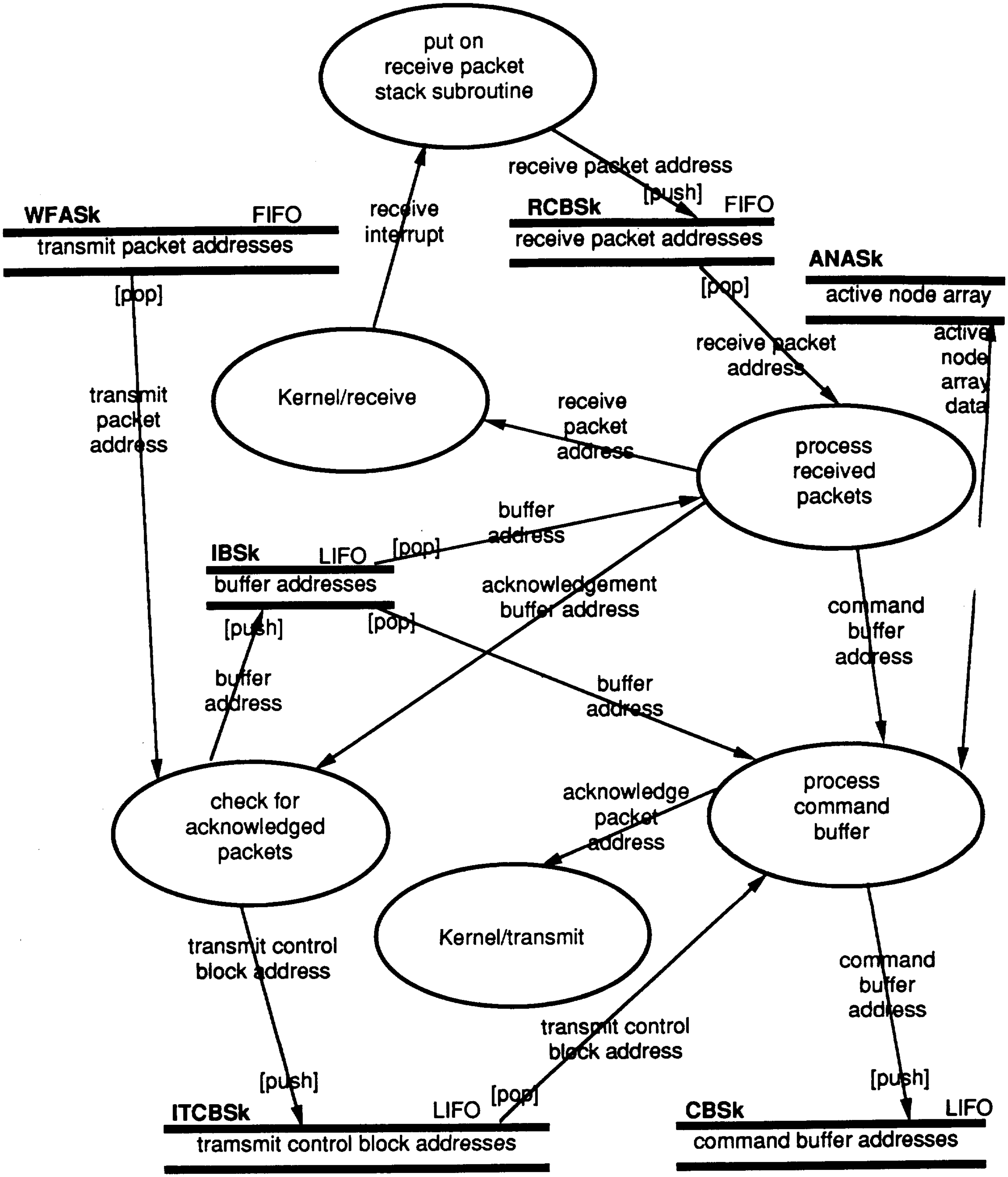


Figure 2 Data Flow Diagram -- Receive Protocol

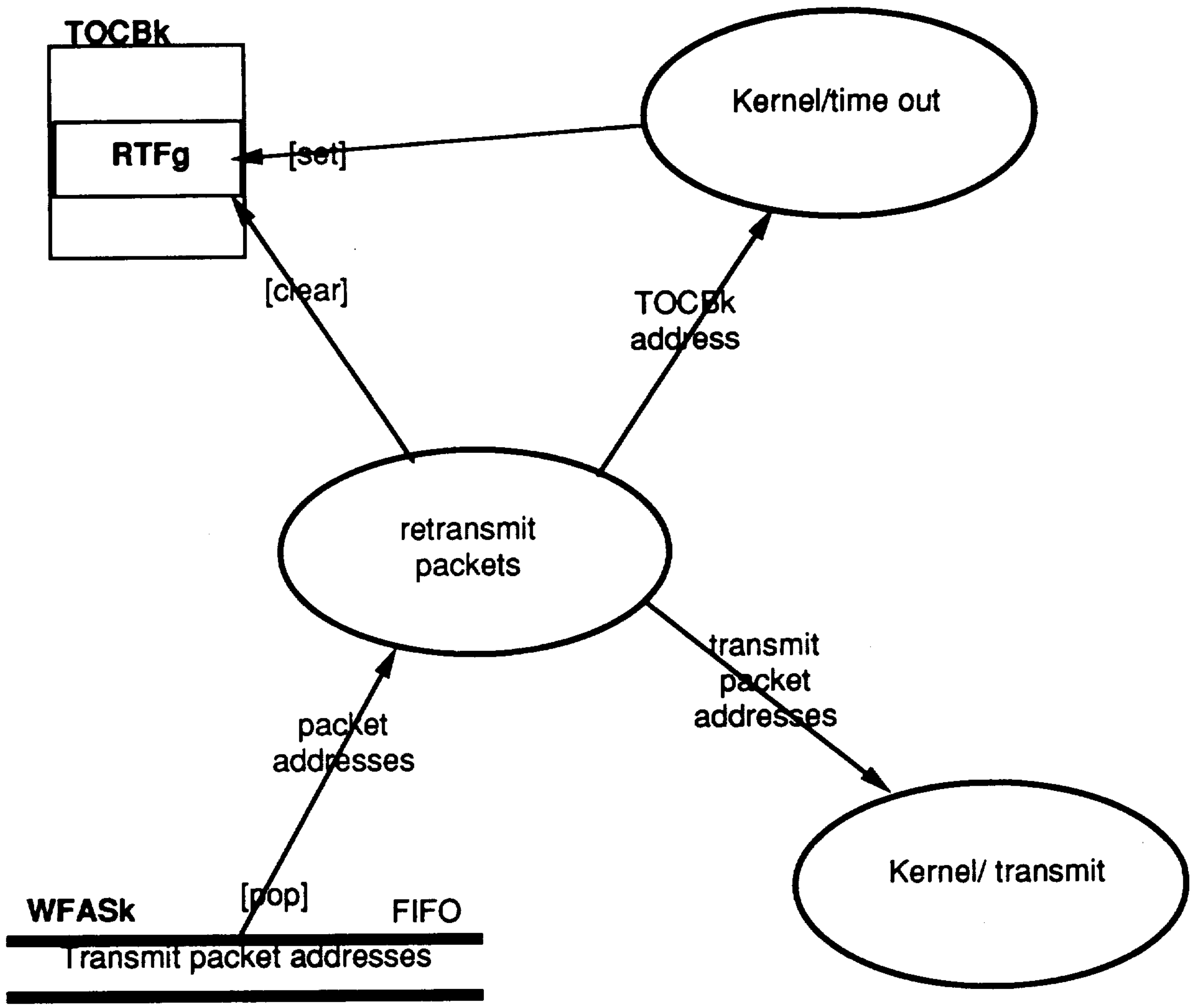


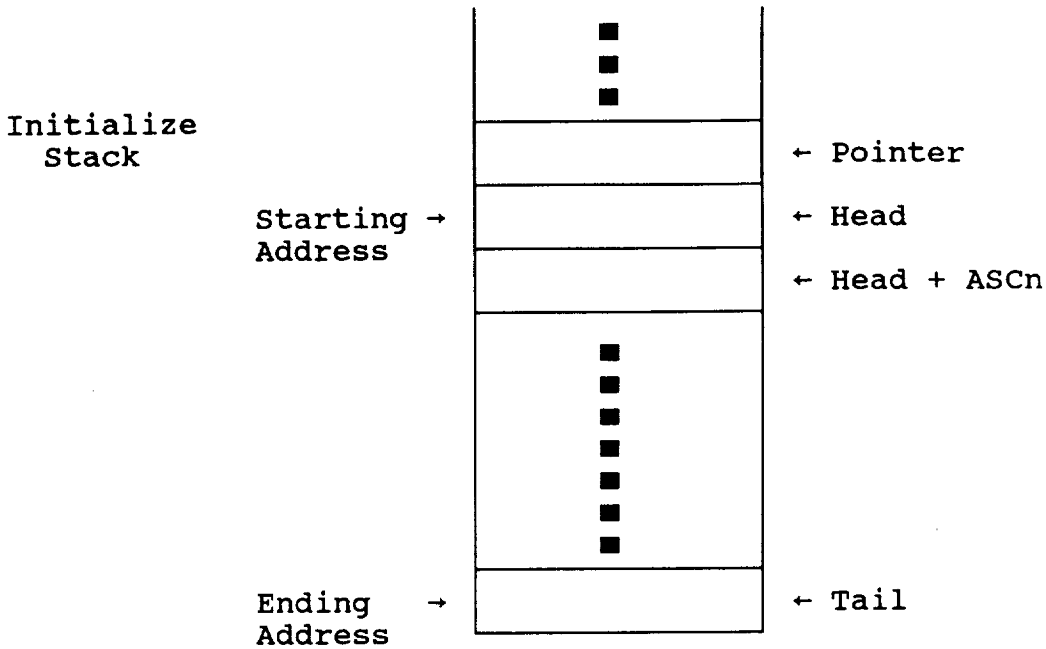
Figure 3 Data Flow Diagram -- Time out Protocol

9. Appendix B Structured Flow Diagrams of Protocol

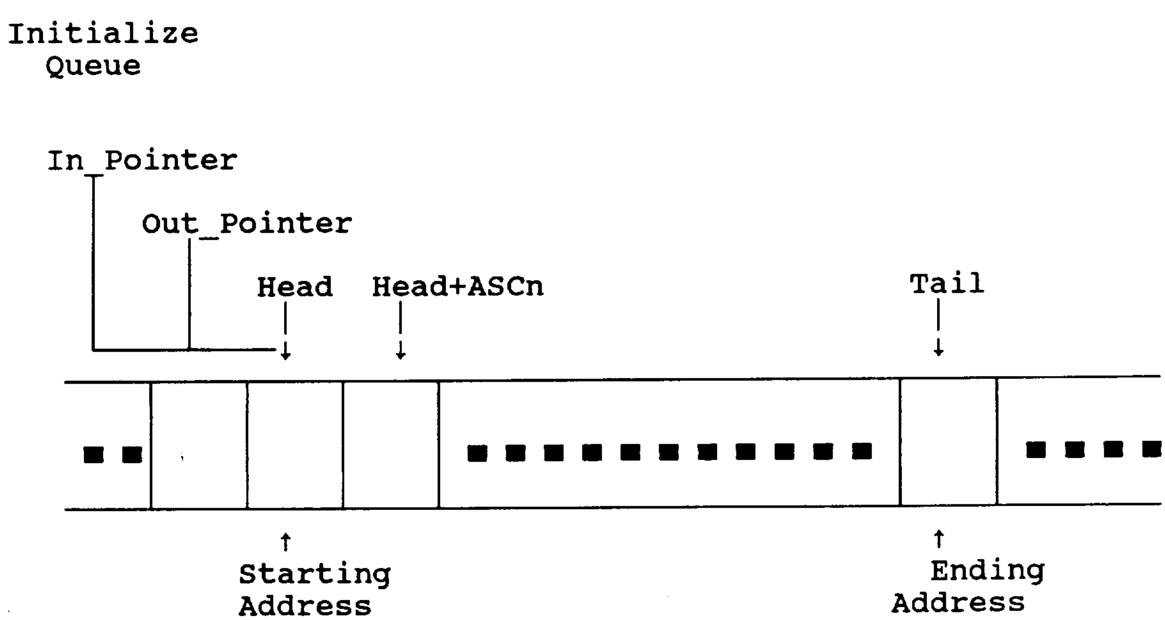
Revision D

9.1. Stack and Queue initialization definitions

9.1.1. Stack



9.1.2. Queue



9.2. Definition of [ ] operator

On the right hand side of the "<-" read "[x]" as: the contents of the location pointed to by x.

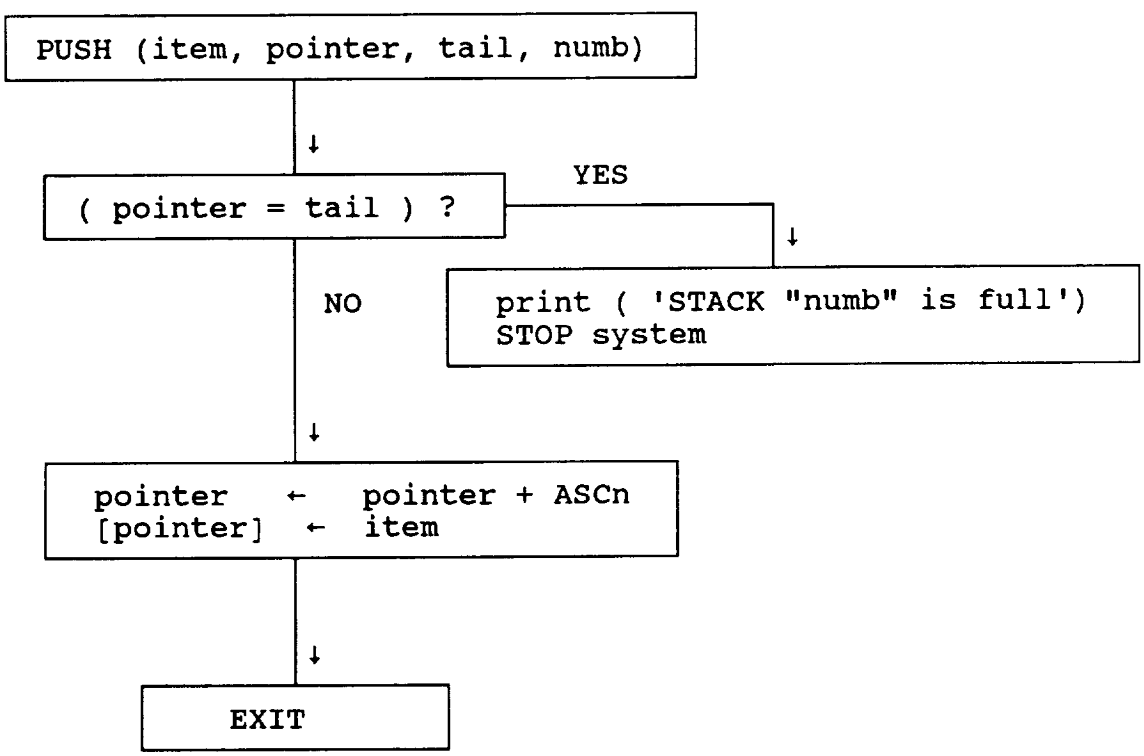
On the left hand side of the "<-" read "[y]" as: store in the location pointed to by y.

9.3. Functions operating on stacks

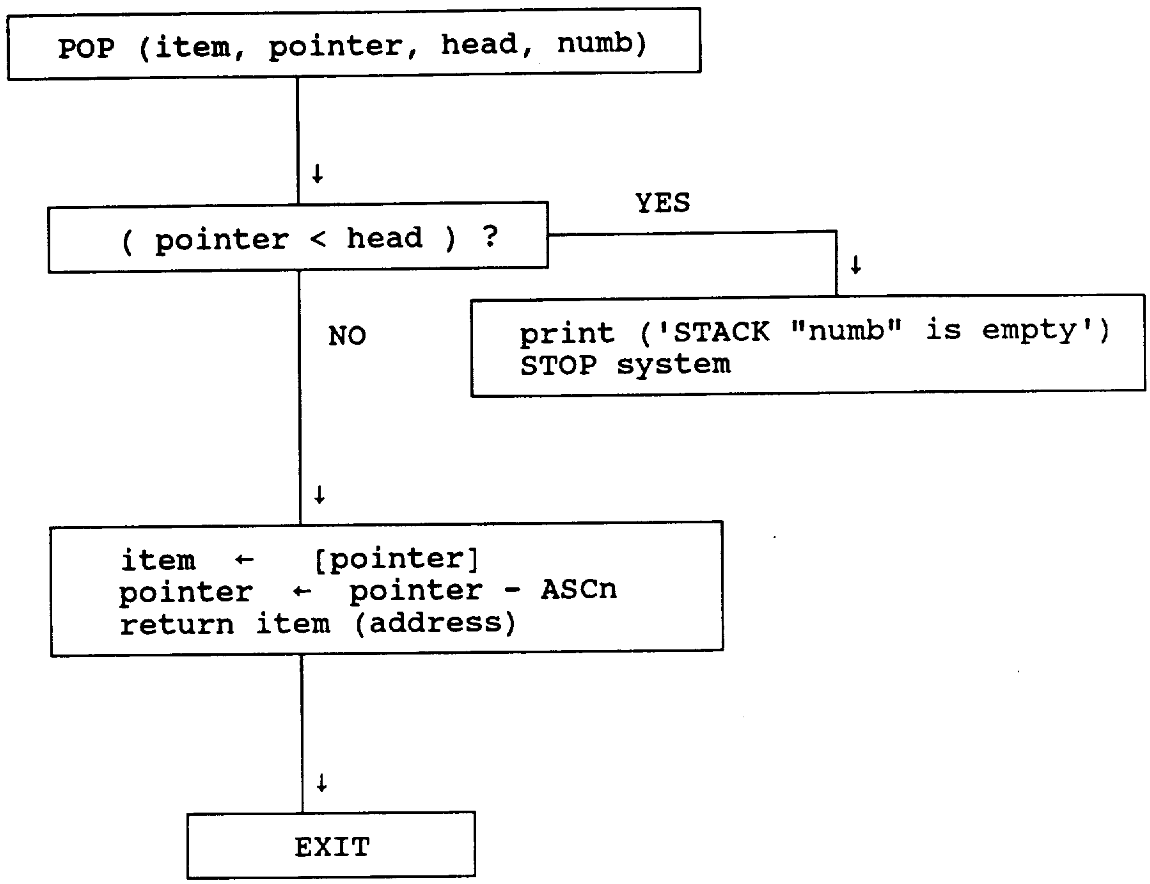
STACK	POINTER	HEAD	TAIL
ITCBSk	ITCBSPp	ITCBSHd	ITCBSTl
IBSk	IBSPp	IBSHd	IBSTl
ANASk	ANASPp	ANASHd	ANASTl

9.3.1. General Push and Pop functions

9.3.1.1. PUSH procedure



9.3.1.2. POP procedure



9.3.2. PUSH_POP_ITCBSk

PUSH

PUSH_ITCBSk (item)

↓

PUSH (item, ITCBSPp, ITCBSTl, 1)

↓

EXIT
procedure call

POP

POP_ITCBSk

↓

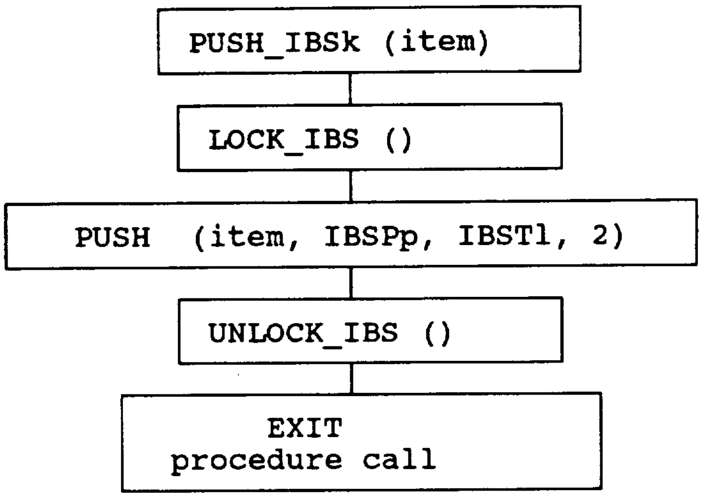
POP (item, ITCBSPp, ITCBSHd, 1)

↓

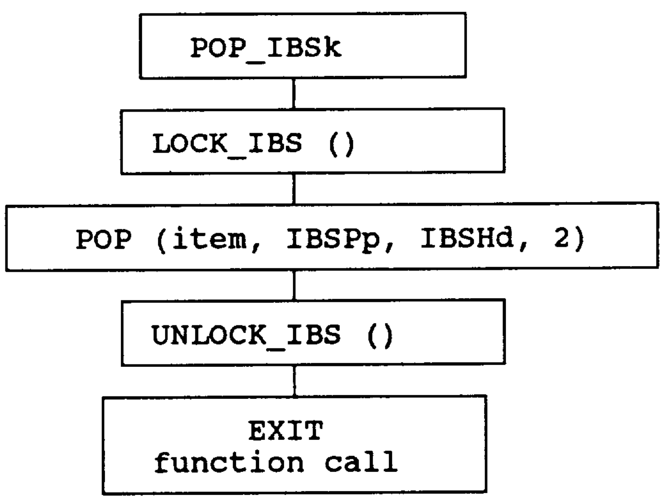
EXIT
function call

9.3.3. PUSH_POP_IBSk

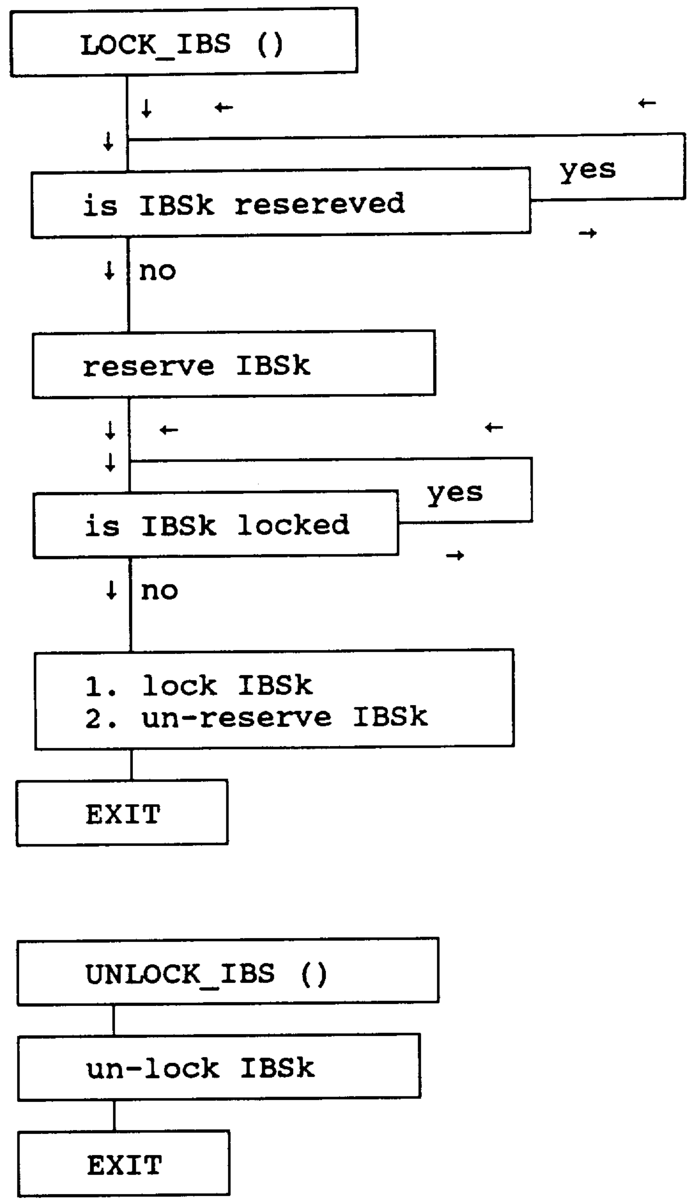
PUSH



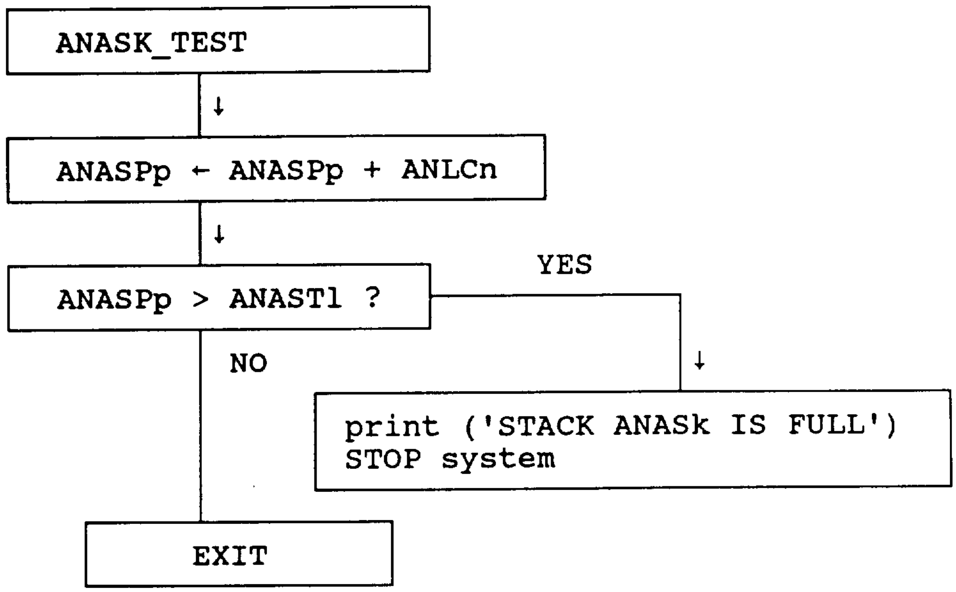
POP

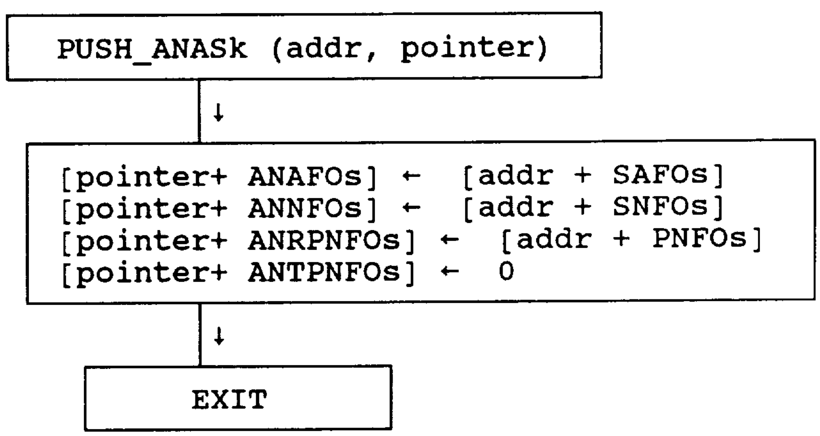


9.3.4. LOCK_IBS and UNLOCK_IBS



9.3.5. PUSH_ANASk

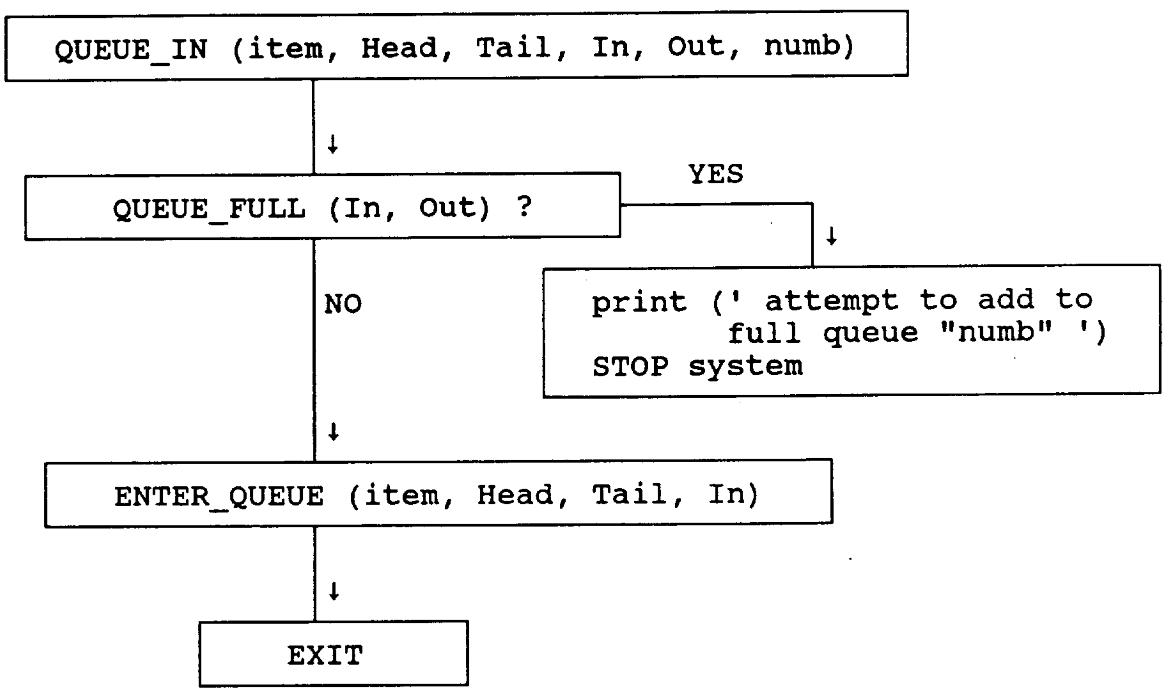




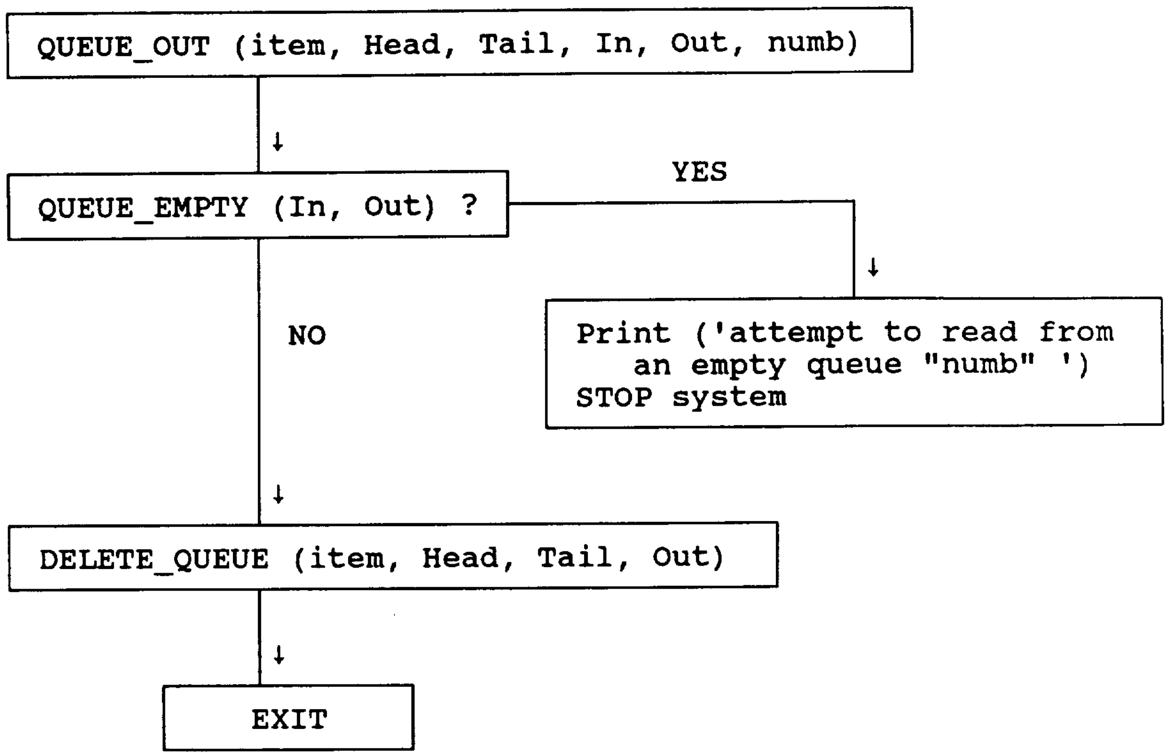
9.4. General operations on Queues

QUEUE	IN-POINTER	OUT-POINTER	HEAD	TAIL
CBSk	CBSIn	CBSOt	CBSHd	CBSTl
RCBSk	RCBSIn	RCBSOt	RCBSHd	RCBSTl
WFASk	WFASIn	WFASOt	WFASHd	WFASTl

9.4.1. Join QUEUE procedure (QUEUE_IN)

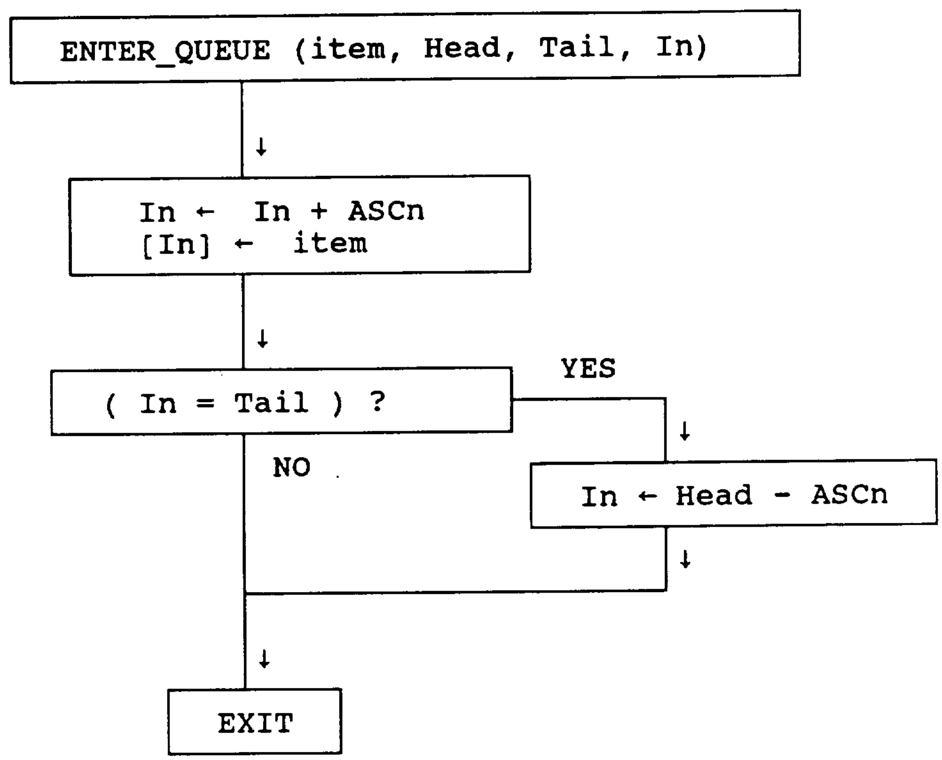


9.4.2. Serve QUEUE procedure (QUEUE_OUT)

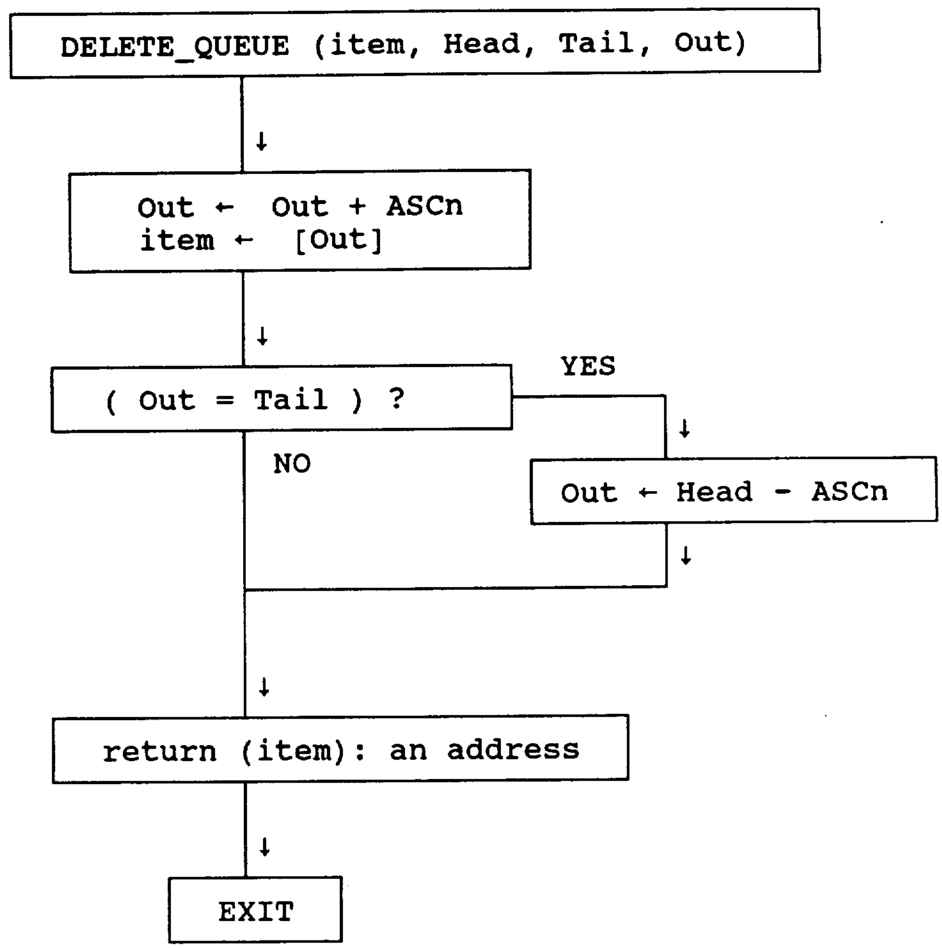


9.4.3. ENTER_QUEUE and DELETE_QUEUE

9.4.3.1. ENTER_QUEUE

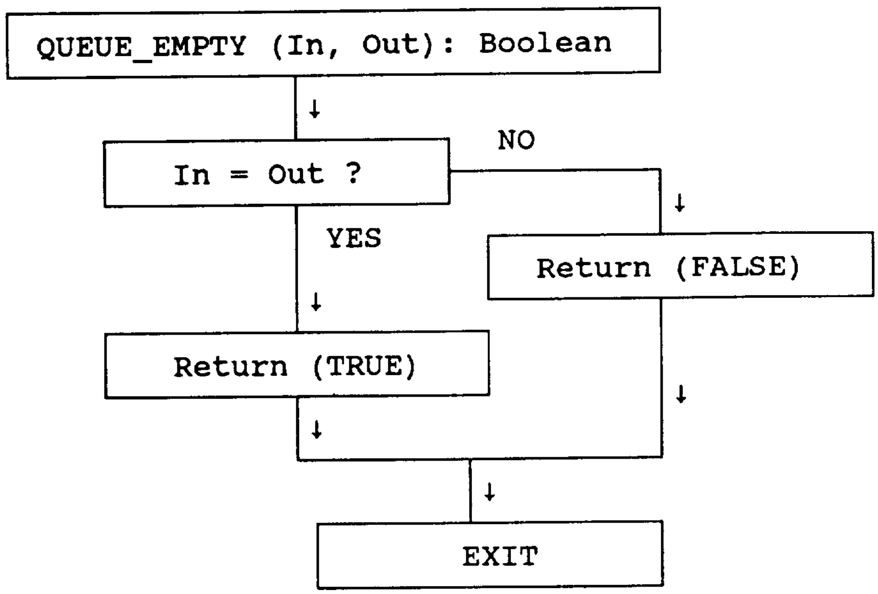


9.4.3.2. DELETE_QUEUE

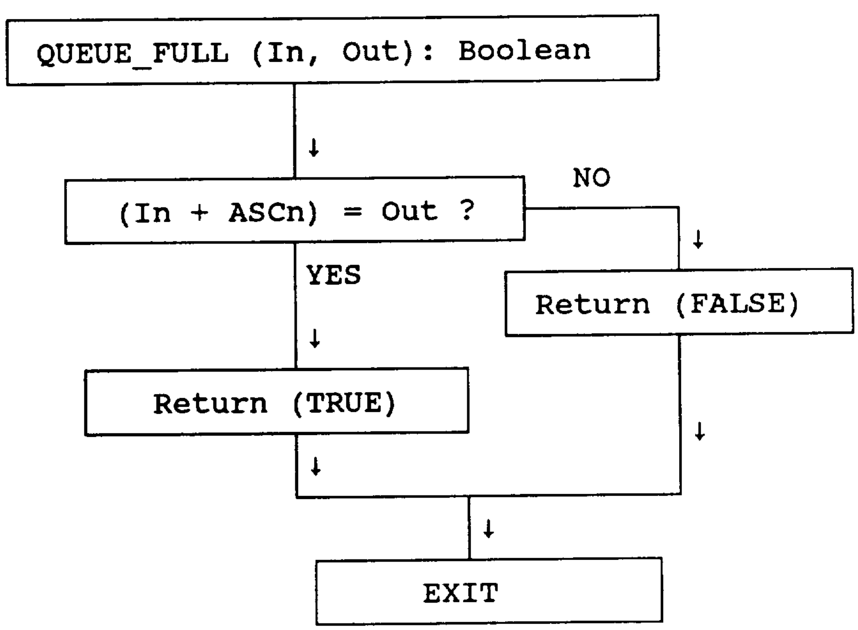


9.4.4. Test CIRCULAR QUEUE Operation

9.4.4.1. EMPTY Function



9.4.4.2. FULL Function



9.4.5. JOIN and SERVE operation on QUEUE

9.4.5.1. QUEUE_IN_OUT_CBSk

Join QUEUE

QUEUE_IN_CBSk (item)

↓

QUEUE_IN (item, CBSHd, CBSTl, CBSIn, CBSOt, 1)

↓

EXIT

Serve QUEUE

QUEUE_OUT_CBSk

↓

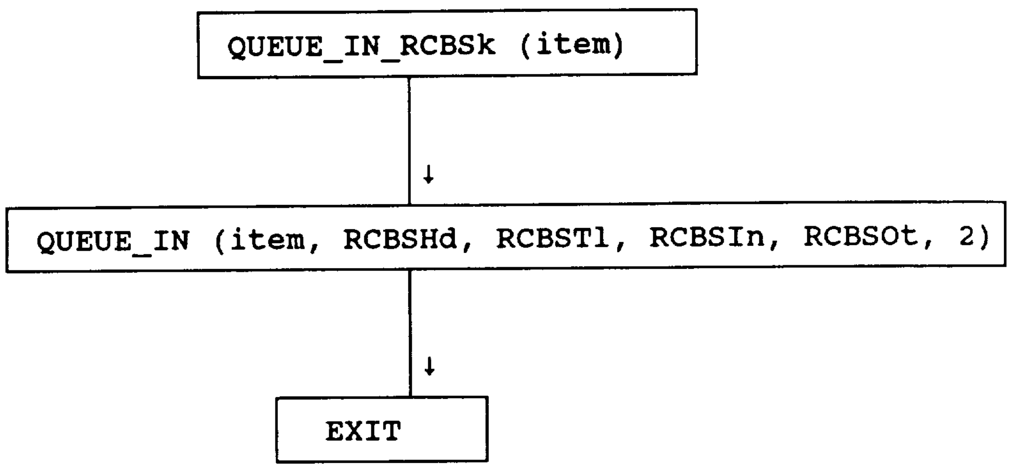
QUEUE_OUT (item, CBSHd, CBSTl,CBSIn, CBSOt, 1)

↓

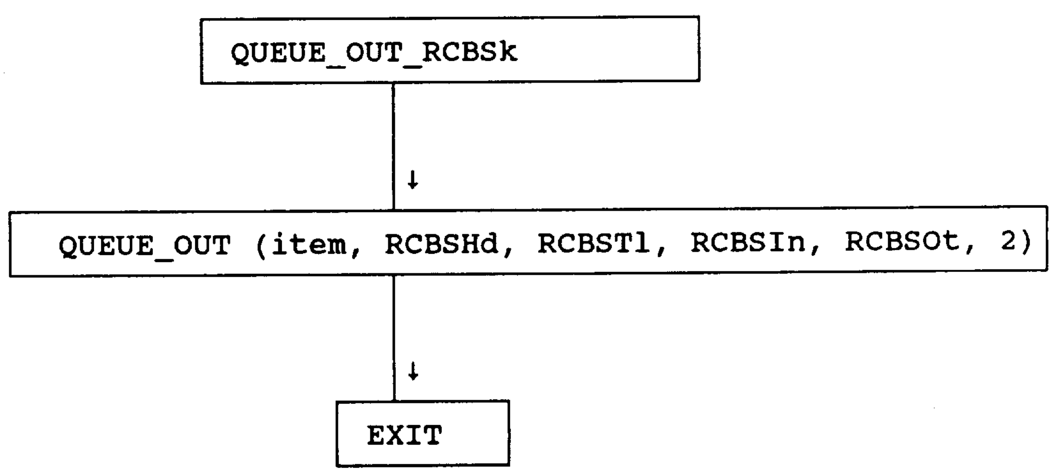
EXIT

9.4.5.2. QUEUE_IN_OUT_RCBSk

Join QUEUE



Serve QUEUE



9.4.5.3. QUEUE_IN_OUT_WFASk

Join QUEUE

QUEUE_IN_WFASk (item)

↓

QUEUE_IN (item, WFAHd, WFASTl, WFASIn, WFASOt, 3)

↓

EXIT

Serve QUEUE

QUEUE_OUT_WFASk

↓

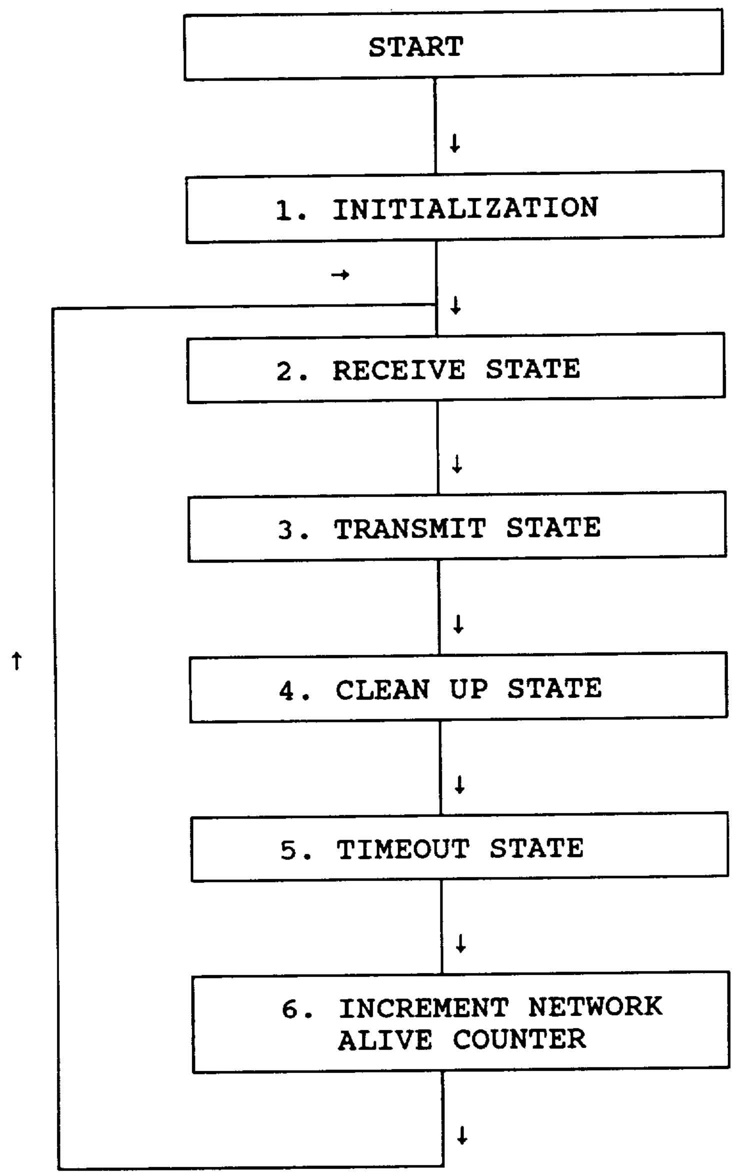
QUEUE_OUT (item, WFASHd, WFASTl, WFASIn, WFASOt, 3)

↓

EXIT

9.5. AMPS Communication Network Structured Flow Diagram

9.5.1. Top level flow diagram



9.5.2. -- Level 1 INITIALIZATION

KRNLINIT ← IAFDDO

↓

1.1 Make STACKS

↓

1.2 Make QUEUES

↓

1.3 Make BUFFERS

↓

1.4 Form ICBk

↓

1.5 Read and Store kernel subroutine addresses

↓

FCFO ← LANCEGO
KSTS (SCBuf)

↓

1.6 call Kernel Receive Ring Fill

↓

SNCn = EPSC ?

YES ↓ 1.7 NETINIT

NO

↓

1.8 Start transmitter timeout

↓

Go To 2. RECEIVE STATE

9.5.2.1. -- Level 1.1 Make STACKS

```
make ITCBSk STACK

BufVr ← BASEADD
ITCBSHd ← BufVr
ITCBSPp ← ITCBSHd - ASCn
BufVr ← BufVr + (ITCBSCn * ASCn)
ITCBSTl ← BufVr - ASCn

        ↓

make IBSk STACK and clear locks

IBSHd ← BufVr
IBSPp ← IBSHd - ASCn
BufVr ← BufVr + (BCn * ASCn)
IBSTl ← BufVr - ASCn
[MAILBOX + IBSKRQOS] ← 0
[MAILBOX + IBSKLKOS] ← 0

        ↓

make ANASk STACK

ANASHd ← BufVr
ANASPp ← ANASHd - ANLCn
BufVr ← BufVr + (ANACn * ANLCn)
ANASTl ← BufVr - ANLCn

        ↓

EXIT
```

9.5.2.2. -- Level 1.2 Make QUEUES

make CBSk QUEUE

```
CBSHd ← BufVr
CBSIn ← CBSHd
CBSOt ← CBSHd
BufVr ← BufVr + (CBSCn * ASCn)
CBSTl ← BufVr - ASCn
```

↓

make RCBSk QUEUE

```
RCBSHd ← BufVr
RCBSIn ← RCBSHd
RCBSOt ← RCBSHd
BufVr ← BufVr + (RCBSCn * ASCn)
RCBSTl ← BufVr - ASCn
```

↓

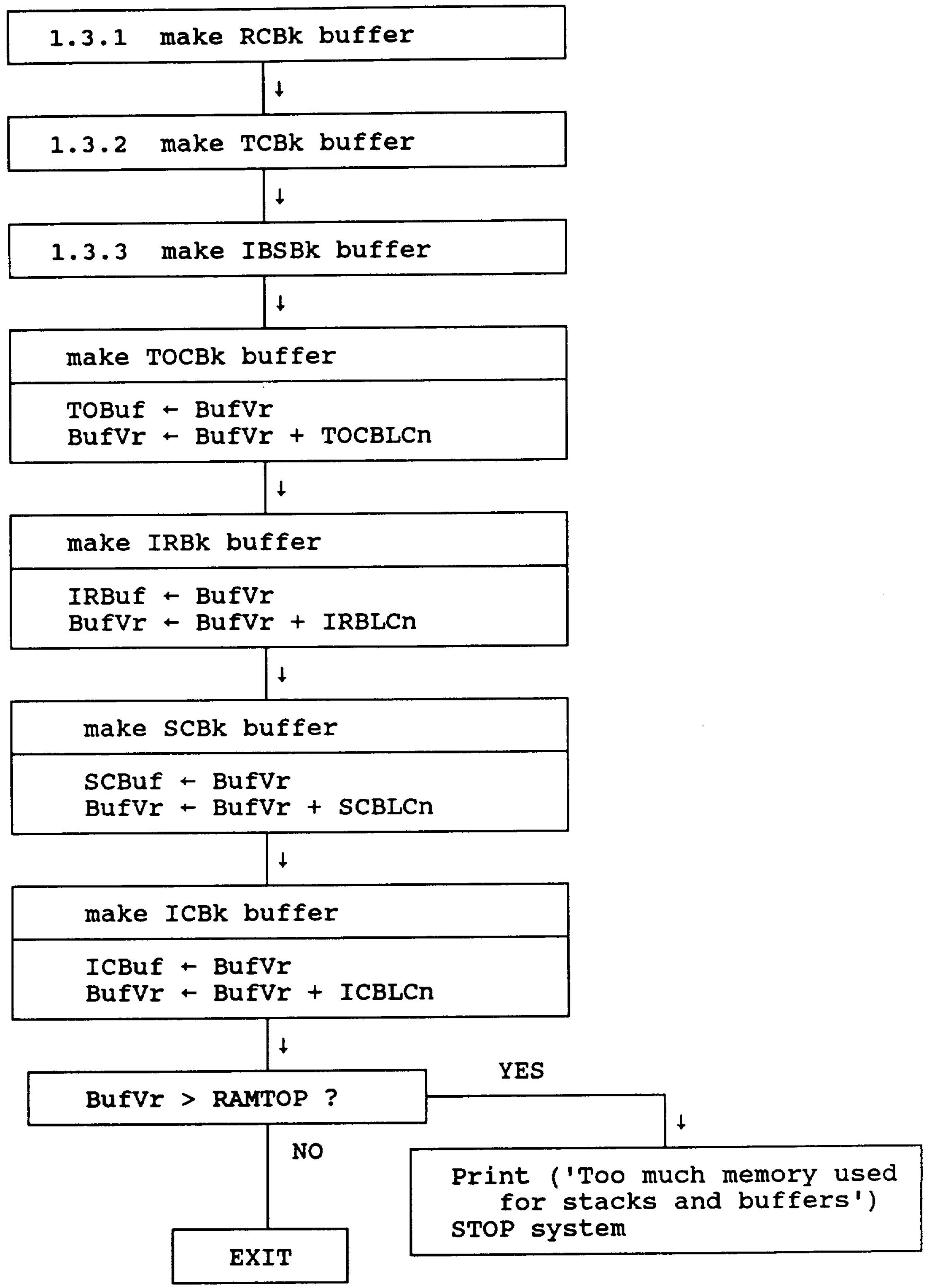
make WFASk QUEUE

```
WFASHd ← BufVr
WFASIn ← WFASHd
WFASOt ← WFASHd
BufVr ← BufVr + (WFASCn * ASCn)
WFASTl ← BufVr - ASCn
```

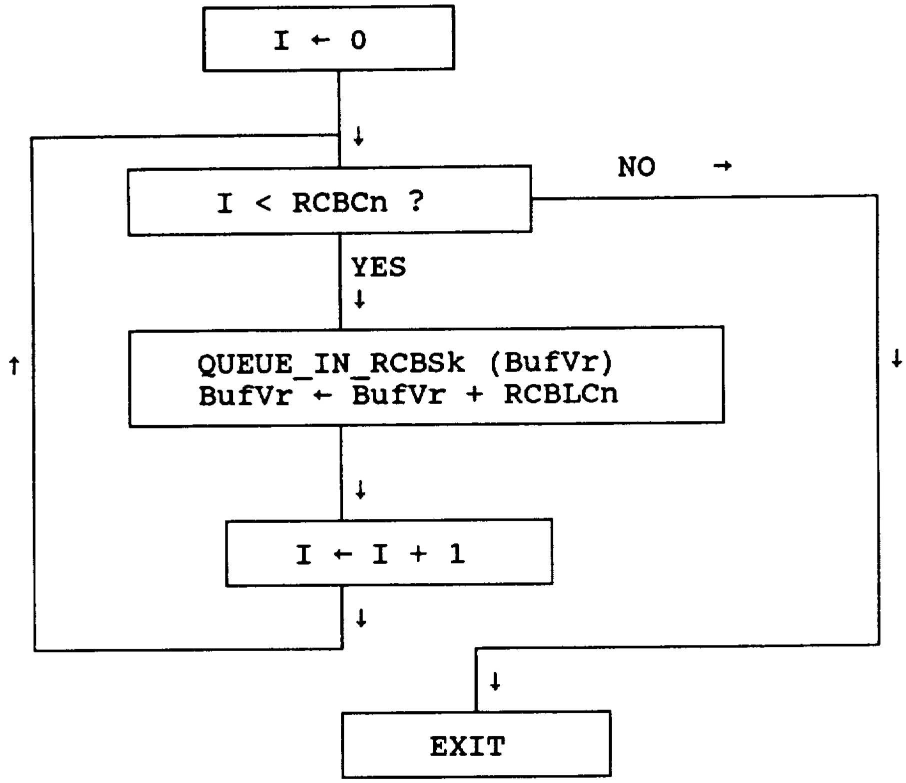
↓

EXIT

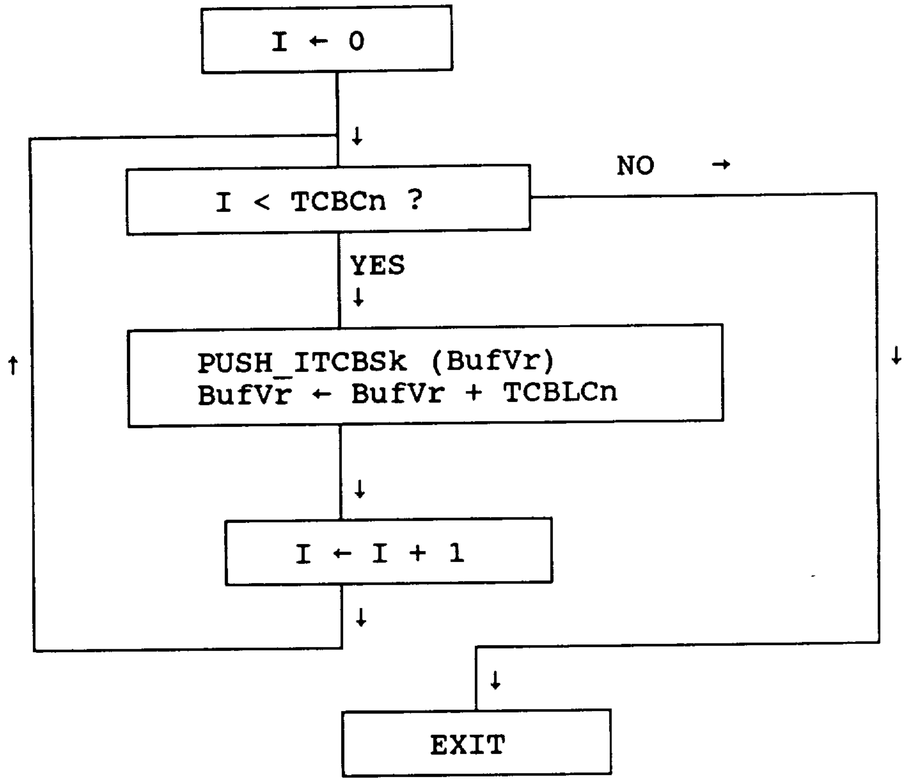
9.5.2.3. -- Level 1.3 Make BUFFERS



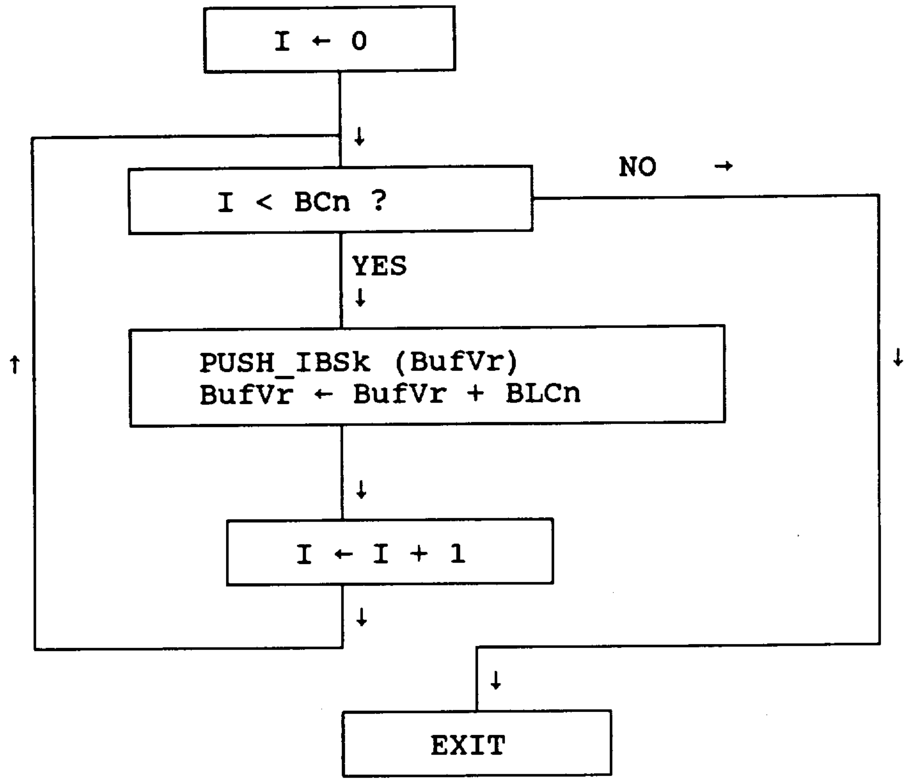
9.5.2.3.1. -- Level 1.3.1 make RCBk buffer



9.5.2.3.2. -- Level 1.3.2 make TCBk buffer



9.5.2.3.3. -- Level 1.3.3 make IBSBk buffer



9.5.2.4. -- Level 1.4 Form ICBk

```
[ICBuf + ICBMFOs] ← LANCMODE
[ICBuf + ICBNRDFOs] ← RCBCn
[ICBuf + ICBNTDFOs] ← TCBCn
[ICBuf + ICBLAFFOs] ← 0
[ICBuf + ICBRIHAFOs] ← RX_INT
[ICBuf + ICBTIHAFOs] ← TX_INT
[ICBuf + ICBBIHAFOs] ← BUS_INT
call KRNLINIT with ICBk on stack (ICBuf, IRBuf)
```

↓

```
EXIT
```

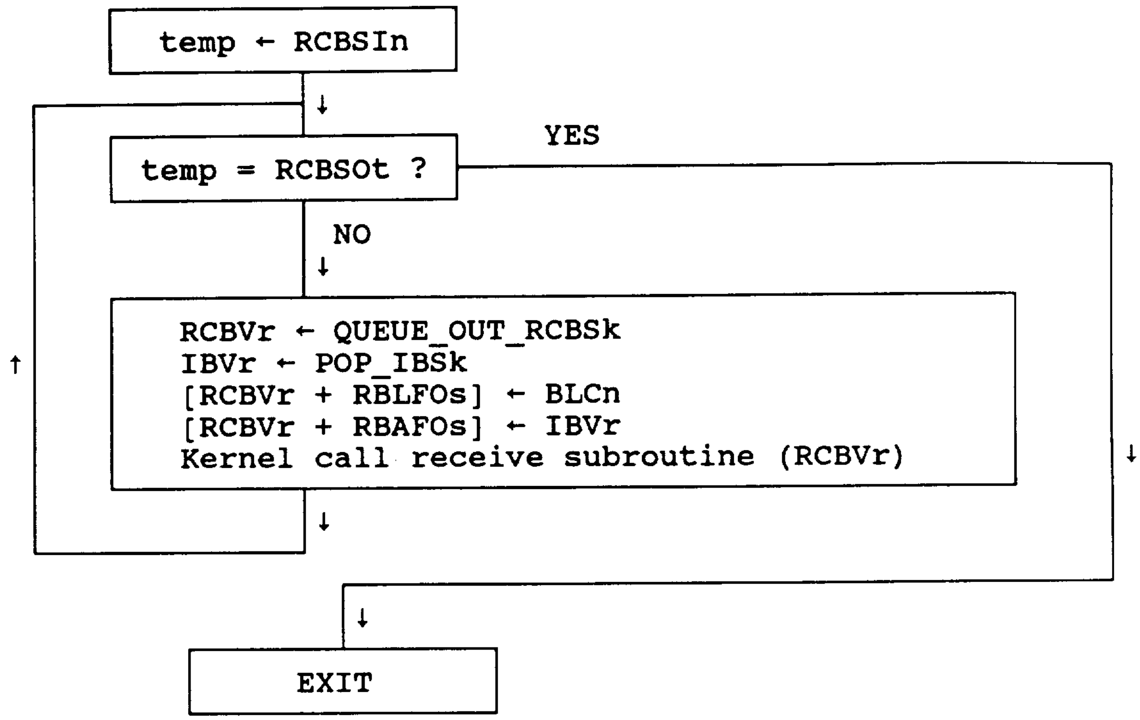
9.5.2.5. -- Level 1.5 Read and Store kernel subroutine addresses

```
SACn ← [IRBuf + IRBESAFOs]
KSTS ← [IRBuf + IRBSRAFOs]
KRCV ← [IRBuf + IRBRRAFOs]
KXMT ← [IRBuf + IRBTRAFOs]
KOUT ← [IRBuf + IRBTORAFOs]
```

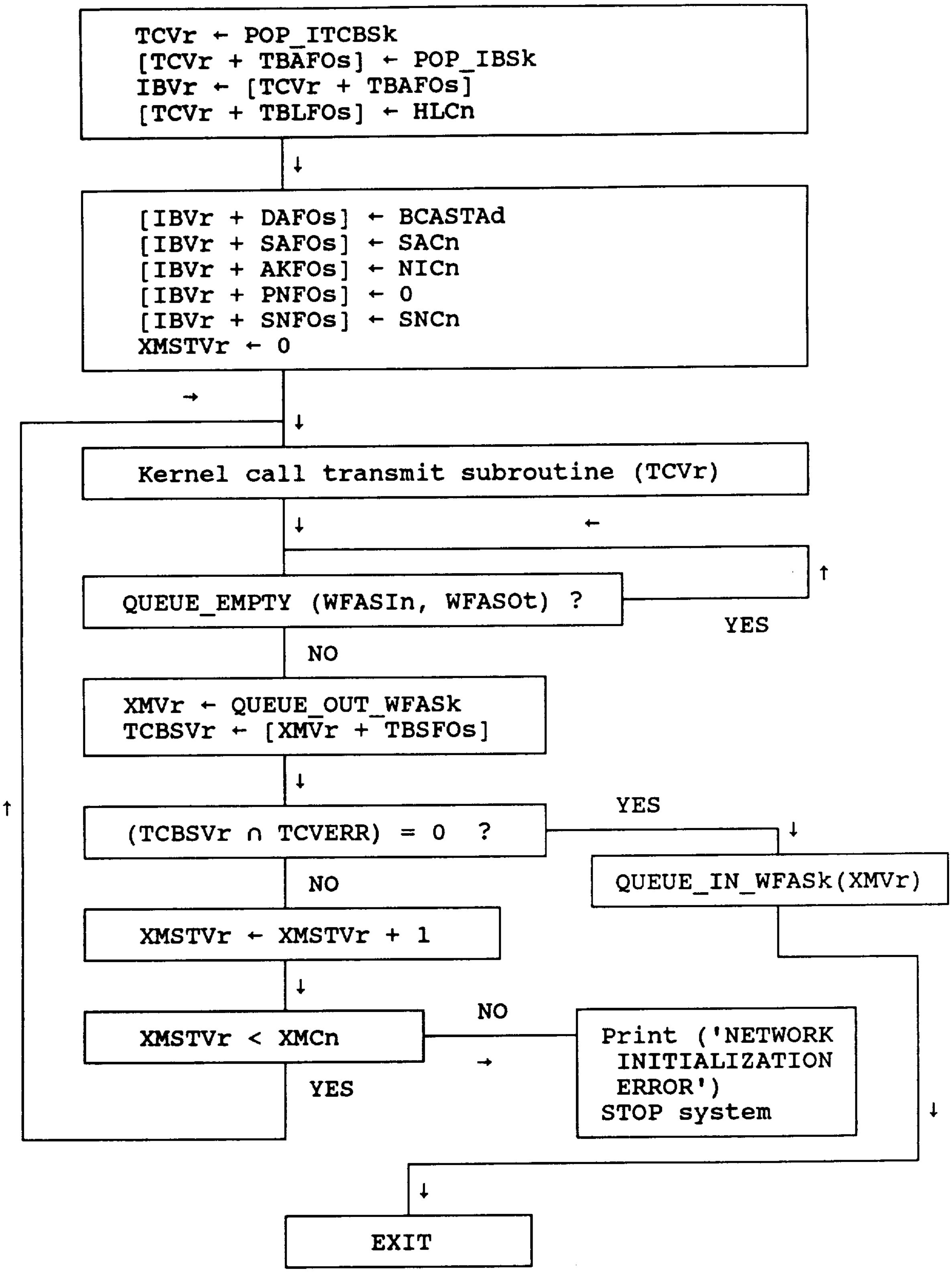
↓

```
EXIT
```

9.5.2.6. -- Level 1.6 call Kernel Receive Ring Fill



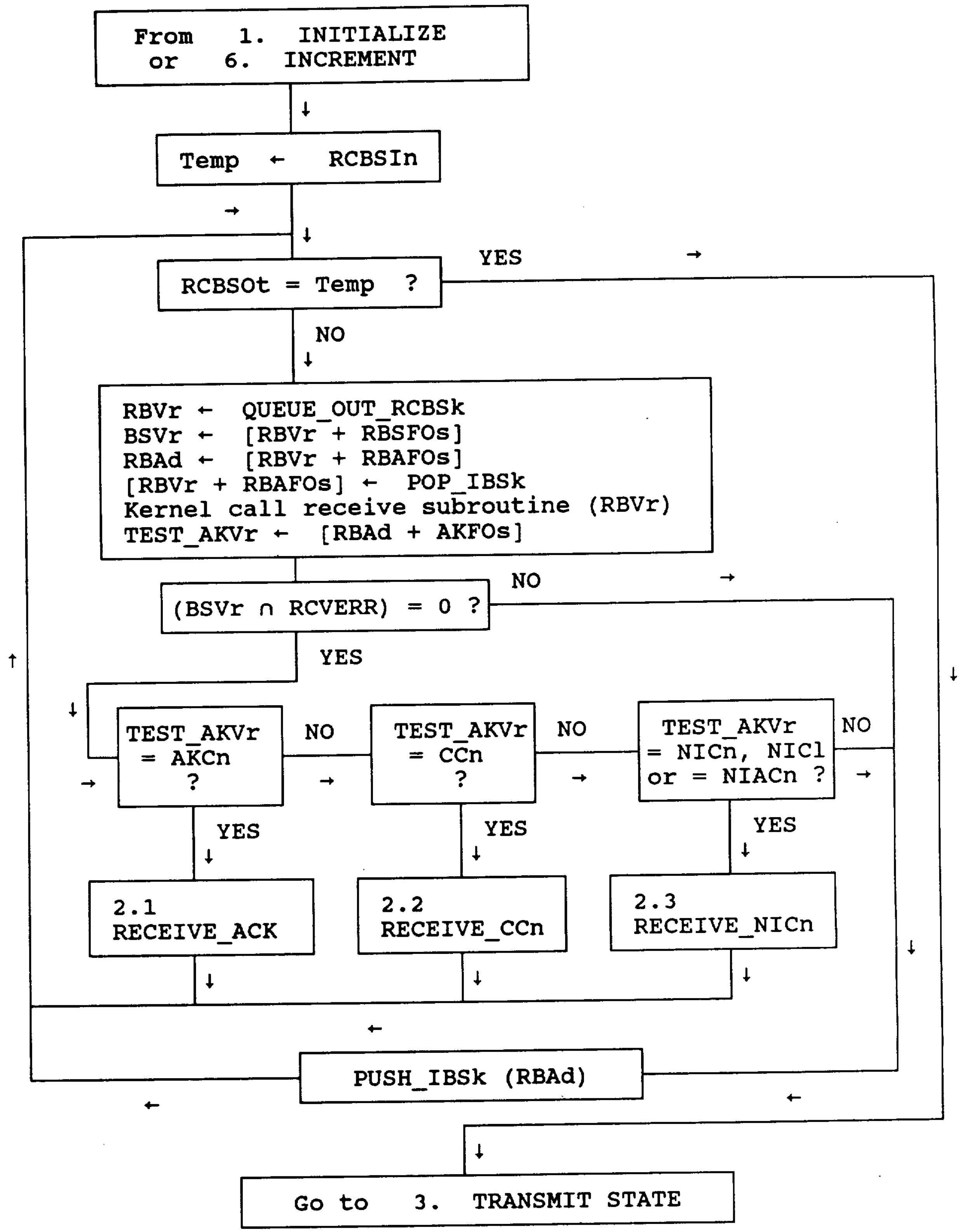
9.5.2.7. -- Level 1.7 KRNLINIT



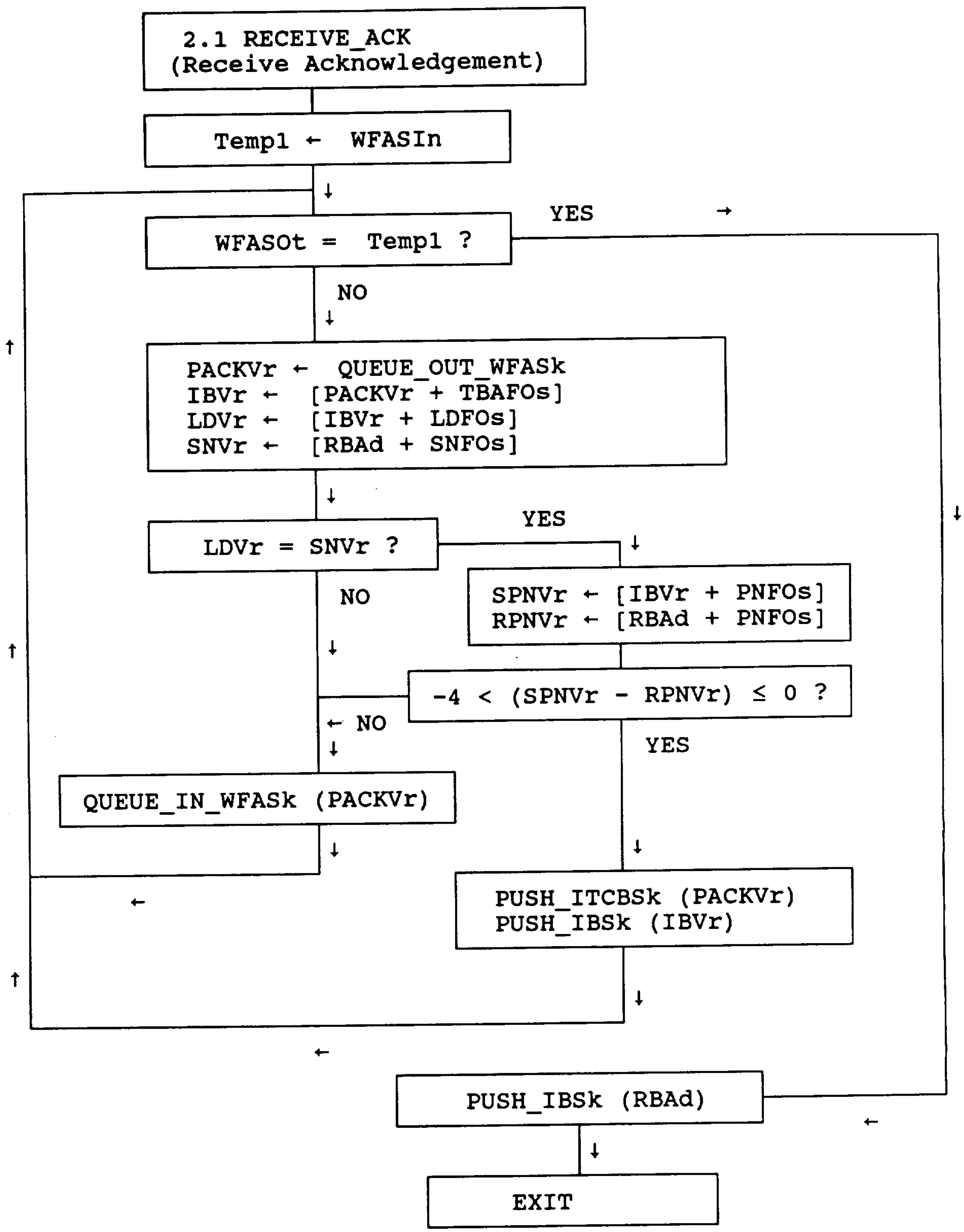
9.5.2.8. -- Level 1.8 Start transmitter timeout

```
[TOBuf + RTFOs] ← TOSET
[TOBuf + TOFOs] ← TOCn
kernel call timeout subroutine (TOBuf)
        ↓
      EXIT
```

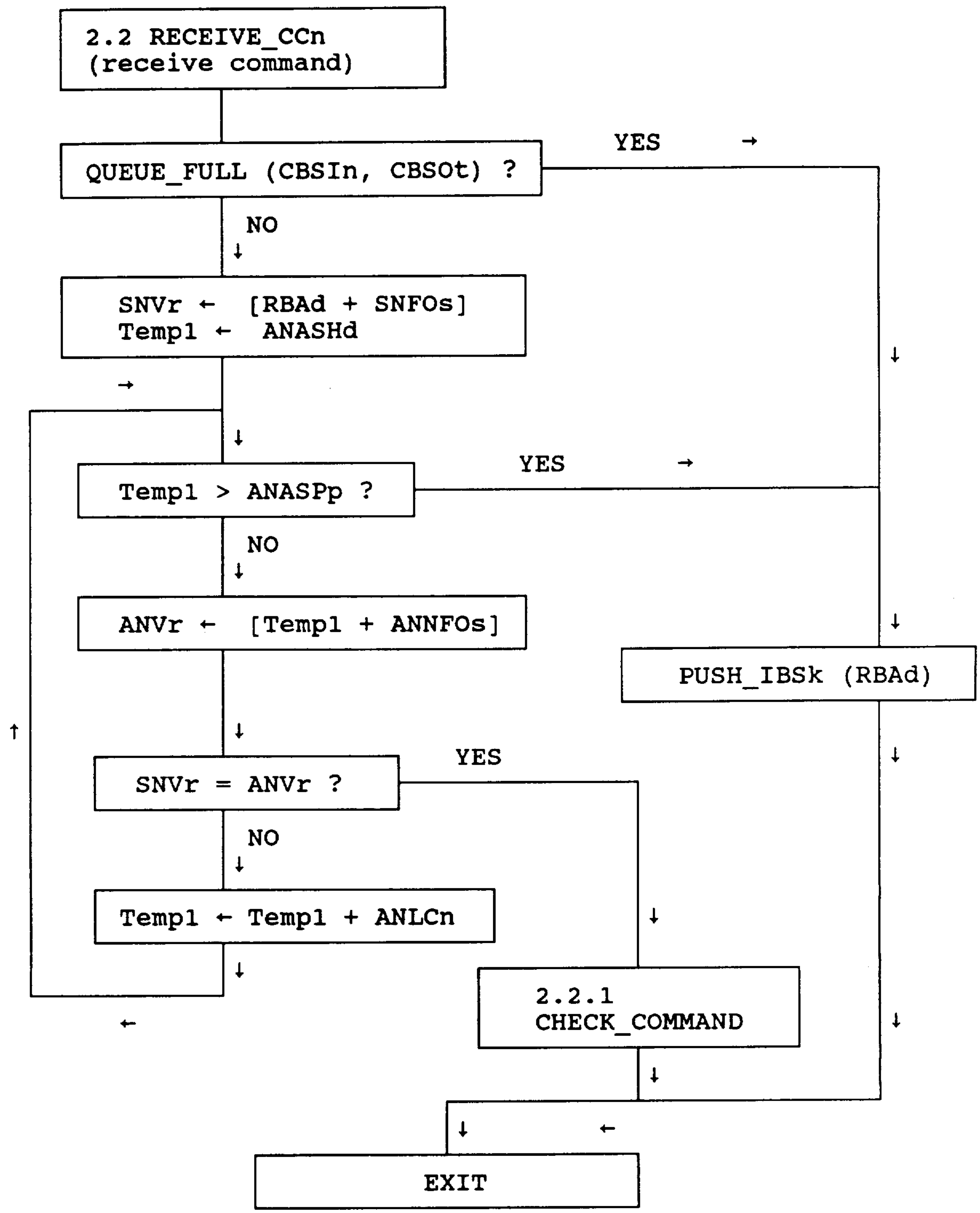
9.5.3. -- Level 2. RECEIVE STATE



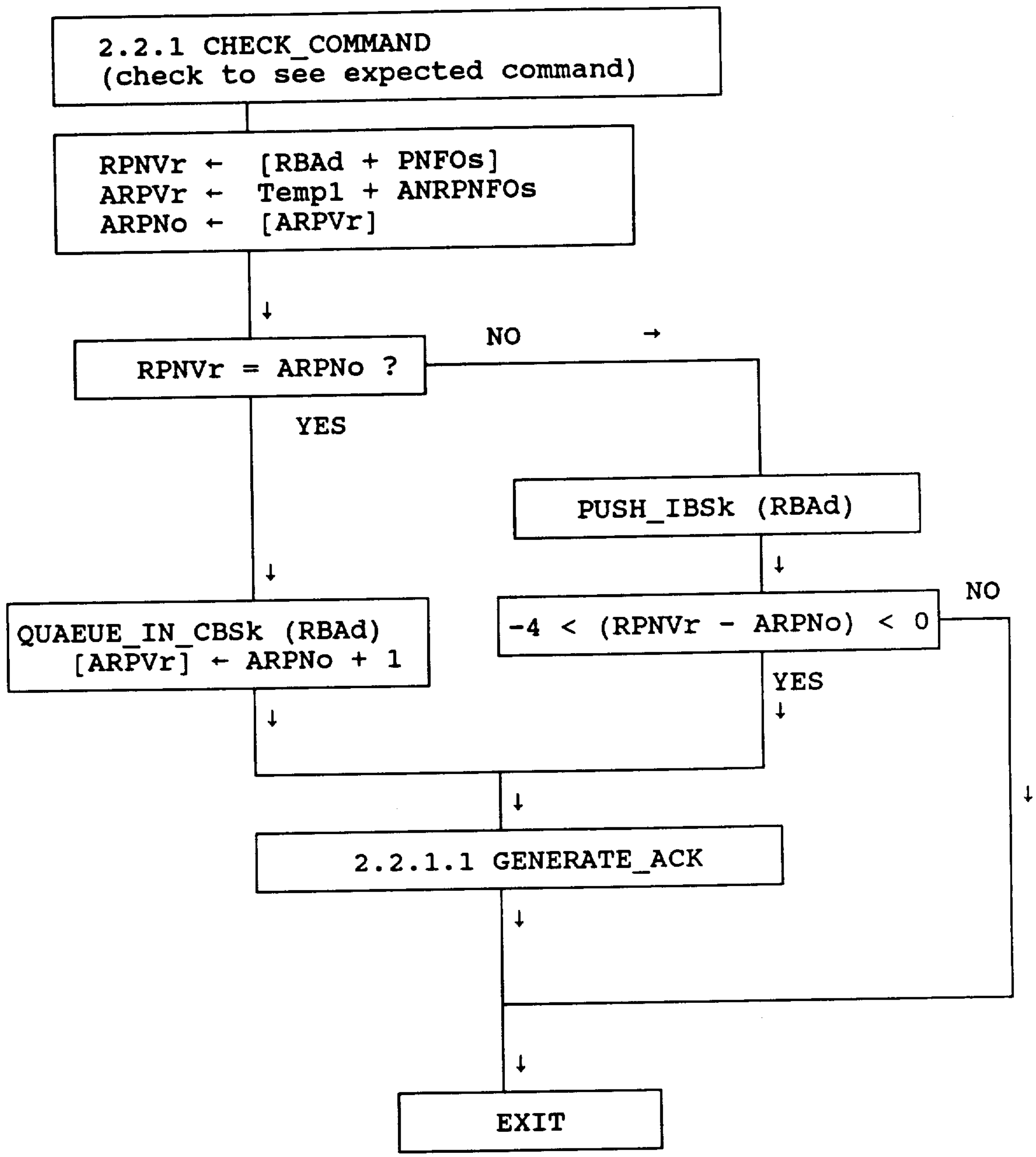
9.5.3.1. -- Level 2.1 RECEIVE_ACK



9.5.3.2. -- Level 2.2 RECEIVE_CCn



9.5.3.2.1. -- Level 2.2.1 CHECK_COMMAND



9.5.3.2.1.1. -- Level 2.2.1.1 GENERATE_ACK

2.2.1.1 GENERATE_ACK

Generate Acknowledgement packet

```
TCVr ← POP_ITCBSk
[TCVr + TBAFOs] ← POP_IBSk
IBVr ← [TCVr + TBAFOs]
[TCVr + TBLFOs] ← HLCn
```

↓

```
[IBVr + DAFOs] ← [RBAd + SAFOs]
[IBVr + SAFOs] ← SACn
[IBVr + AKFOs] ← AKCn
[IBVr + PNFOs] ← [RBAd + PNFOs]
[IBVr + SNFOs] ← SNCn
```

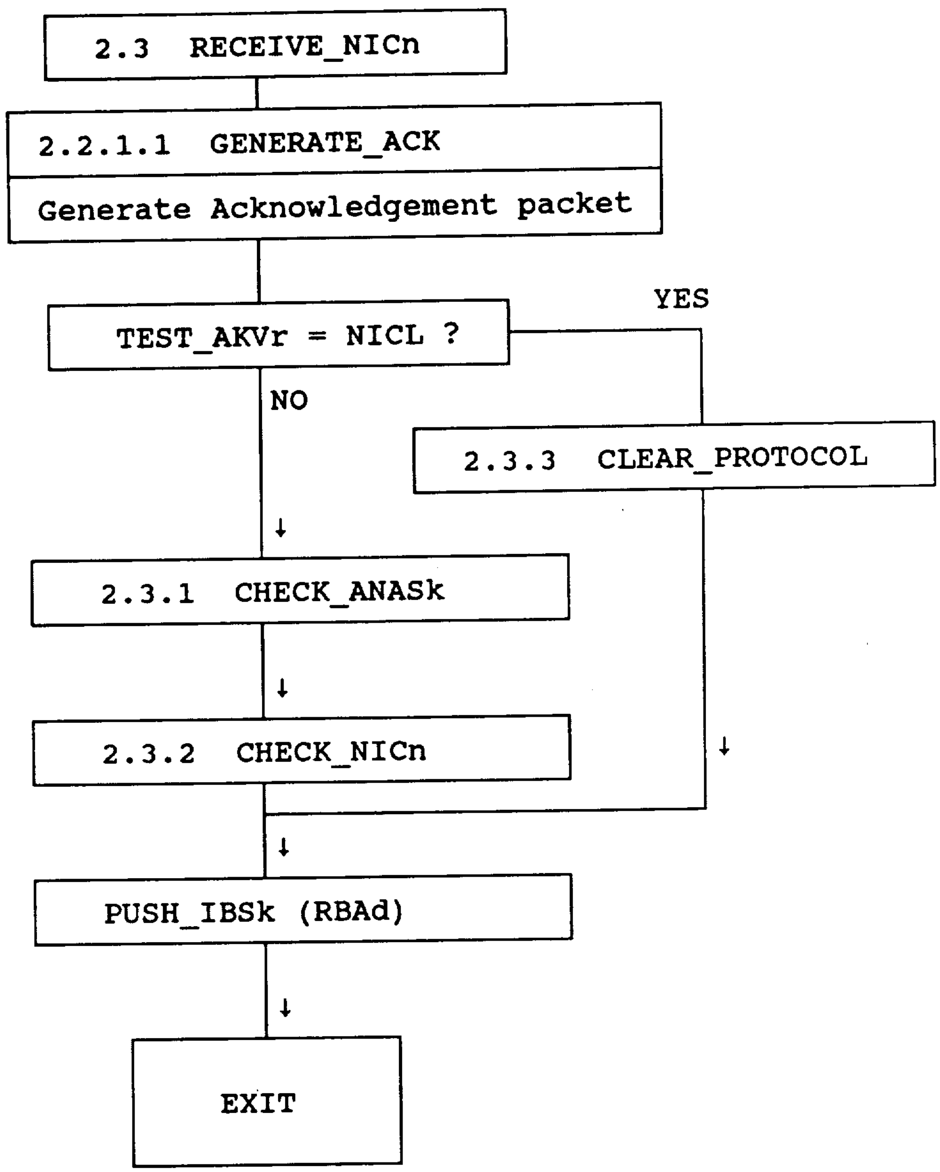
↓

Kernel call transmit subroutine (TCVr)

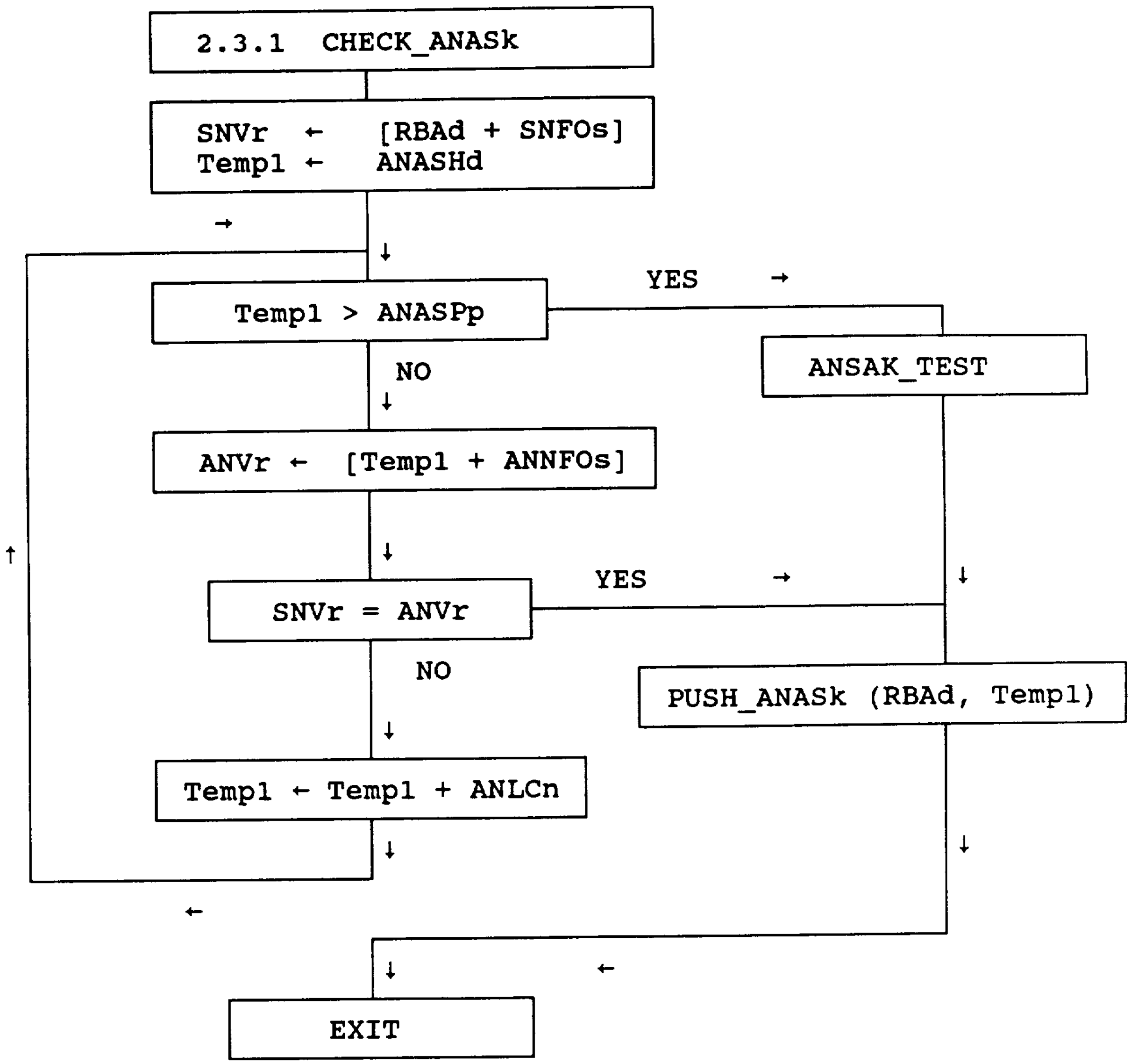
↓

EXIT
GENERATE_ACK procedure

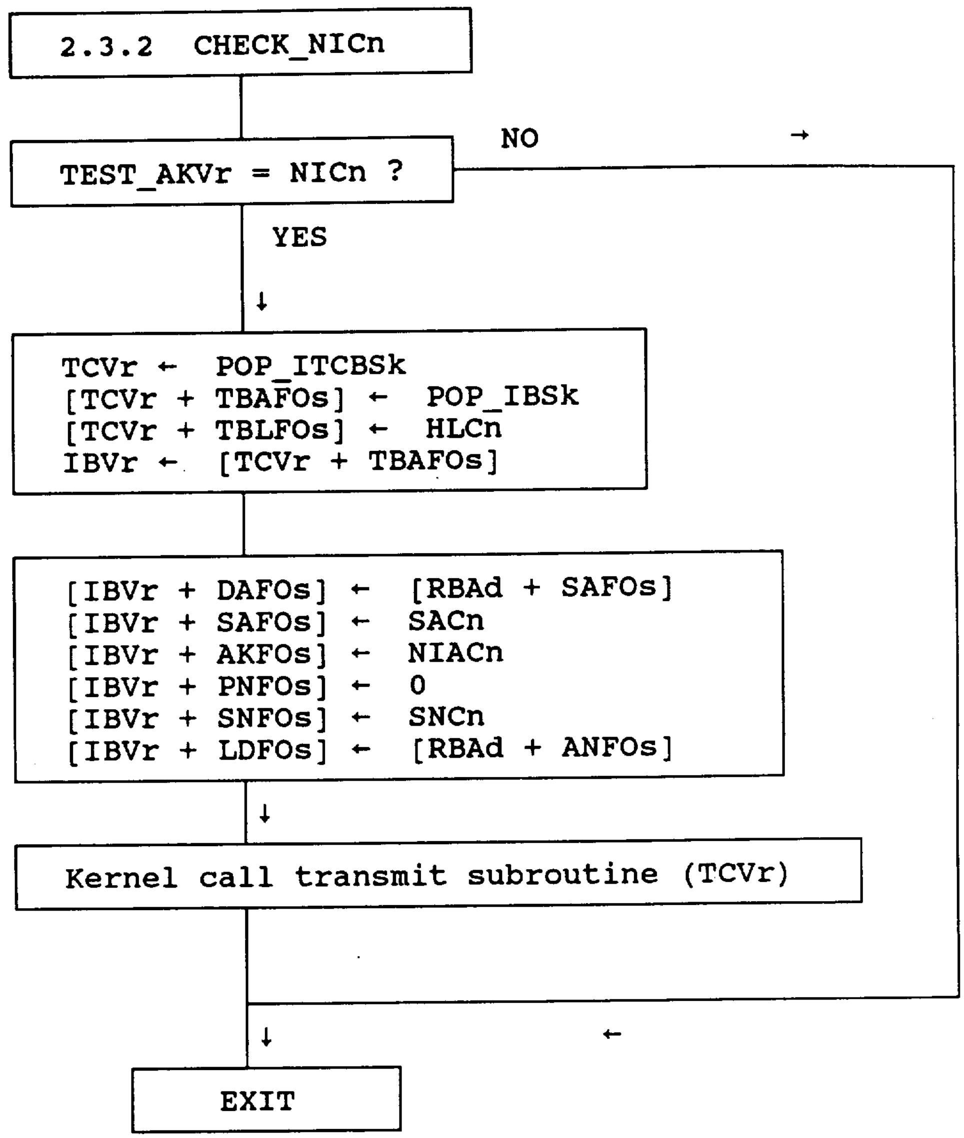
9.5.3.3. -- Level 2.3 RECEIVE_NICn



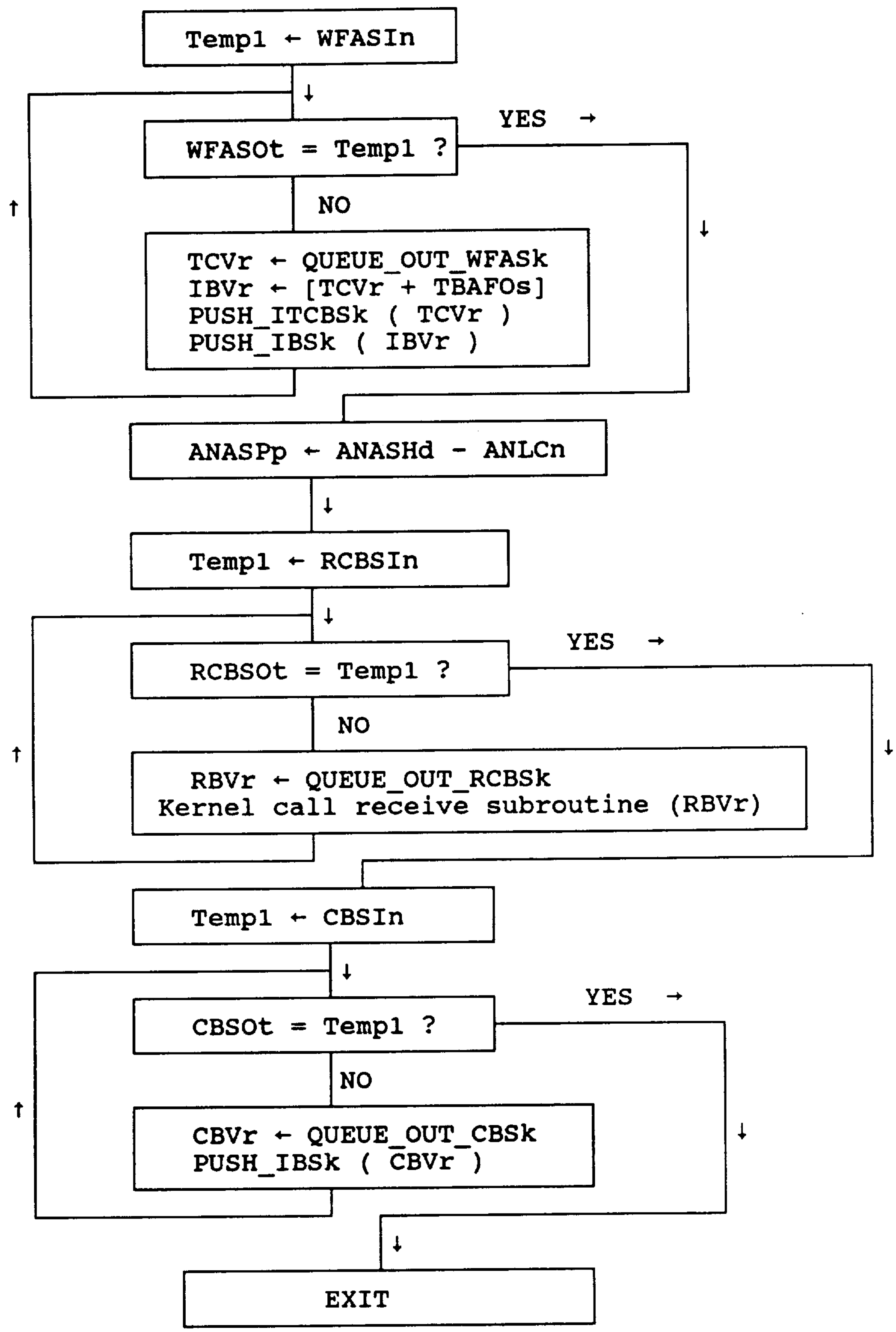
9.5.3.3.1. -- Level 2.3.1 CHECK_ANASk



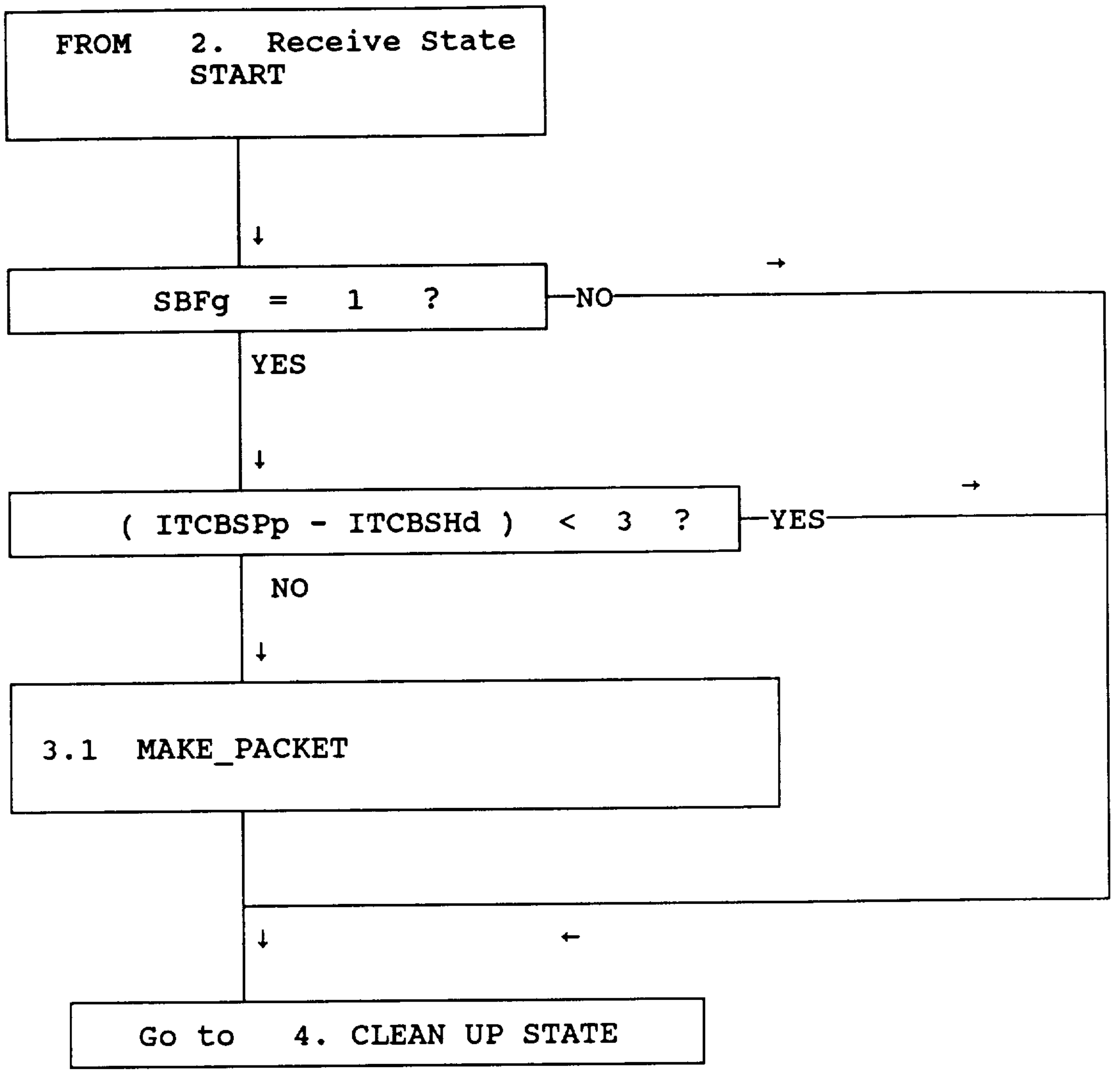
9.5.3.3.2. -- Level 2.3.2 CHECK_NICn



9.5.3.3.3. -- Level 2.3.3 CLEAR_PROTOCOL



9.5.4. -- Level 3. TRANSMIT STATE



9.5.4.1. -- Level 3.1 MAKE_PACKET

3.1 MAKE_PACKET

↓

TCVr ← POP_ITCBSk
[TCVr + TBAFOs] ← SBAd
PACKVr ← [TCVr + TBAFOs]
[TCVr + TBLFOs] ← [PACKVr + DLFOs] + HLCn
SBFg ← 0

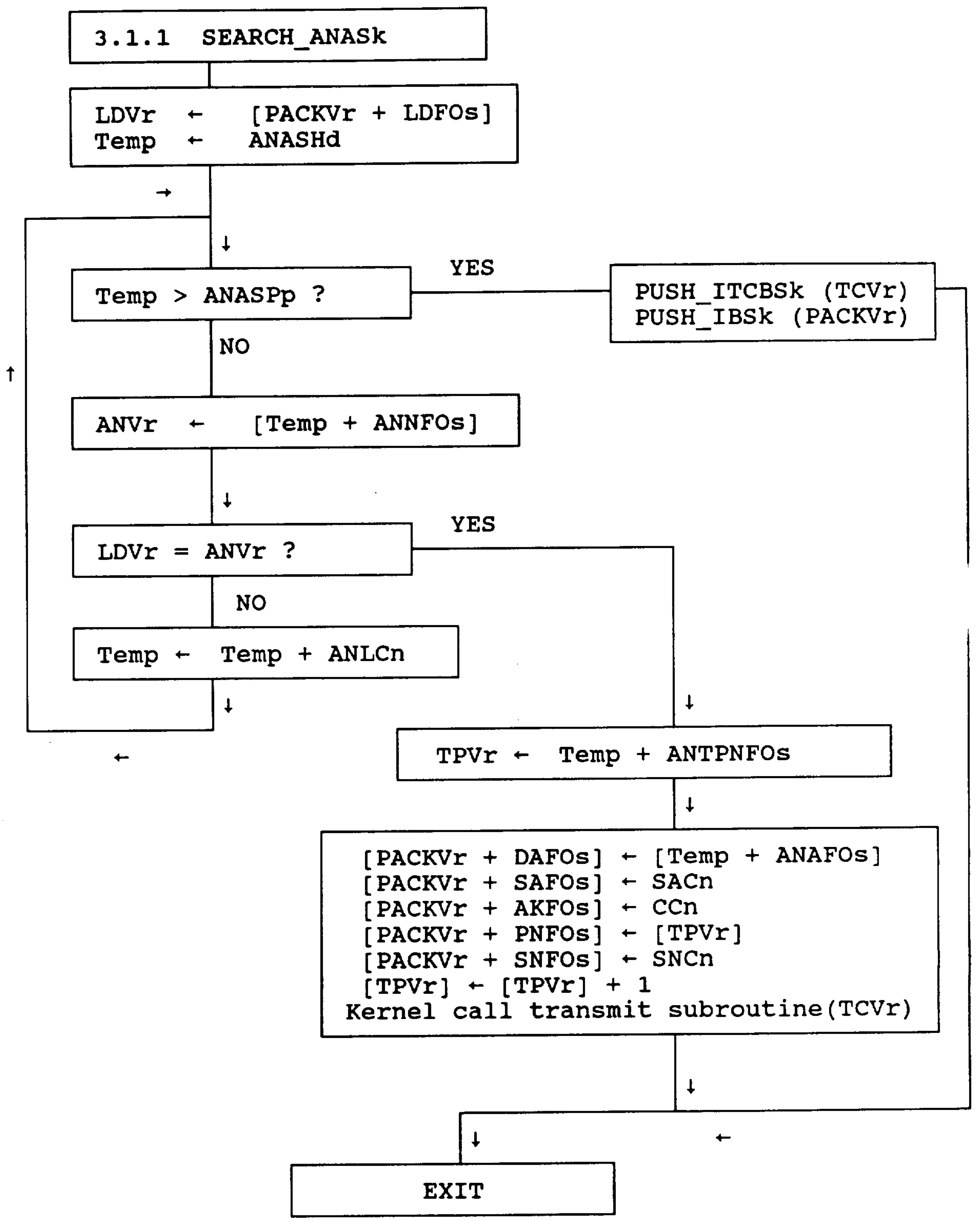
↓

3.1.1 SEARCH_ANASk

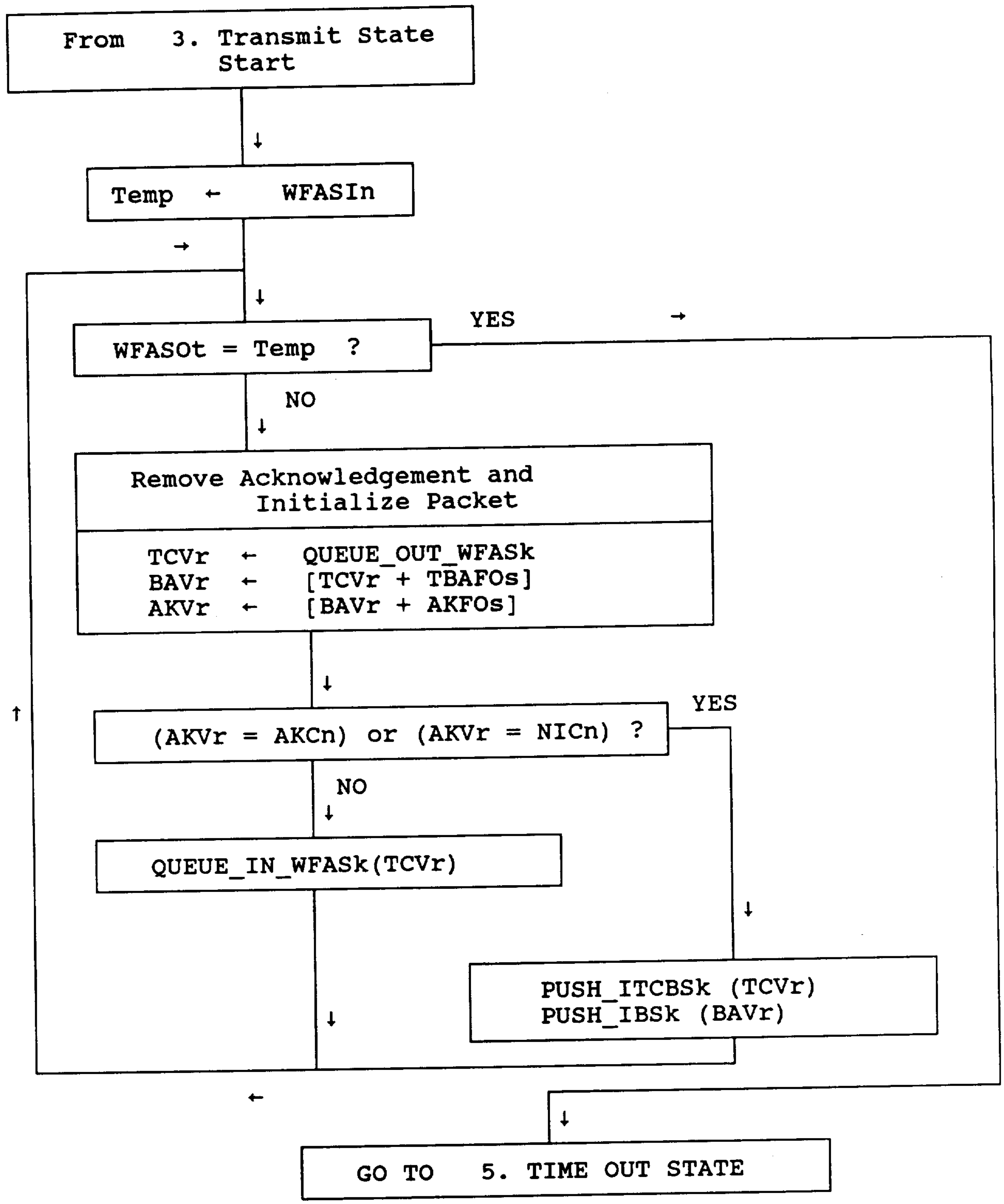
↓

EXIT

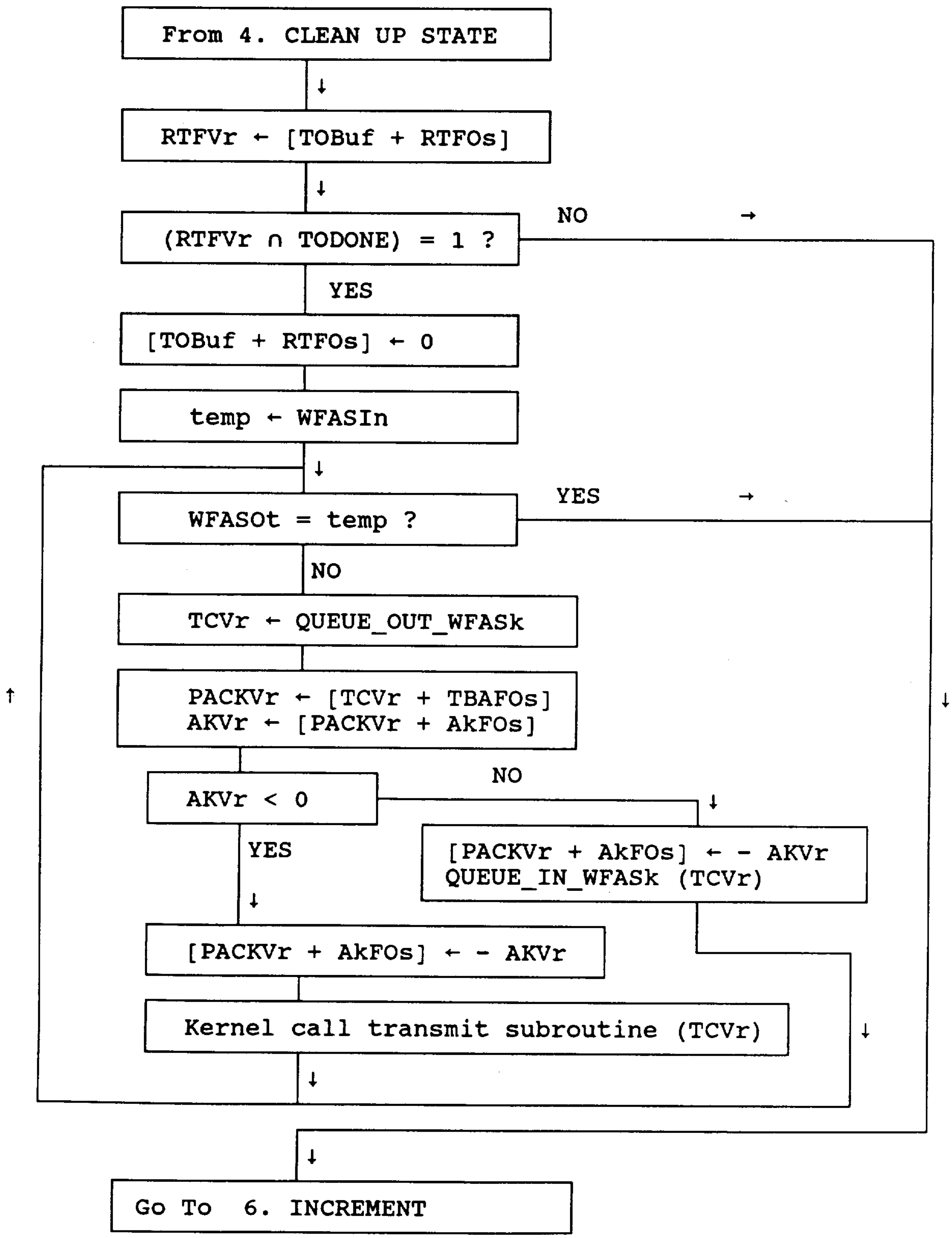
9.5.4.1.1. -- Level 3.1.1 SEARCH_ANASk



9.5.5. -- Level 4. CLEAN UP STATE



9.5.6. -- Level 5. TIMEOUT STATE



10. Appendix C Protocol source code in FOURTH

```
1299 LIST

( Compiler)    HEX
FORTH : IMMEDIATE   8000 PREVIOUS  NAME 4- CFA 4+ NAME - -
    TW+!  VOC W@  0003 VOC W!  TARGET VOC W! ;
FORTH : [COMPILE]   FORTH  CONTEXT W@  0003 CONTEXT W!
    '   EXECUTE  CONTEXT W! ;    TARGET   HOST DECIMAL
| CODE ?ALLOT ( n - t)    H U) DO MOV   S )+ DO ADD   1 #U DO ADD
    0 # DO BCLR   DO D1 MOV   250 # D1 ADD   D1 S CMP
    CS NOT IF DO H U) MOV   ' EXIT BRA   THEN   I S -) MOV NEXT
: ALLOT ( n)    ?ALLOT ABORT" dictionary full" ;   RECOVER
CODE ?CELL    S ) DO MOV   32768 # DO ADD   -65536 # DO AND
    DO S -) MOV   NEXT
: C, ( n)    HERE C!  1 H +! ;    : , ( n)    4 ALLOT  HERE 4- ! ;
: W, ( n)    2 ALLOT  HERE 2- W! ;
CODE COMPILE    H U) AO MOV   1 )+ AO )+ MOV   AO H U) MOV  NEXT
: LITERAL ( n)    ?CELL IF COMPILE long  , ELSE
    COMPILE cell W,  THEN ;    IMMEDIATE

1300 LIST

1301 LIST
```


ORIGINAL PAGE IS
OF POOR QUALITY

1302 LIST

```
( Target Compiler for ENP-30)    EMPTY  DECIMAL
: HELP  1304 HELPS ;    ( HELP)   HEX 1010 WIDTH W! DECIMAL
SHADOWS 20 - CONSTANT NEW          1306 CONSTANT ENP30
: AHEAD ( - a)    >IN W@  32 WORD   SWAP >IN W! ;
: LOG ;    : .WHERE ;     110 LOAD
CREATE HEAD  32 ALLOT   CREATE HDS  4 ALLOT   CREATE ?TWO  1 W,
VARIABLE WO    VARIABLE 'H    VARIABLE 'R   WVARIABLE VOC
( Changed to allow for 32k ENP30 application DW UAH)
( Meta)  46 LOAD   HOST DEFINITIONS
: ORG ( a)    1+ 2/ 2*  'H ! ;     : HERE ( - a)    'H @ ;
: GAP ( n)    HERE + ORG ;     : THERE ( - a)    'R @ ;
HEX    : WINDOW ( n)   DUP ORG WO !  0 'R !  000B VOC W!
             HEAD 20 ERASE ;    DECIMAL
: !    0 HDS W! ;          : RECOVER    -4 GAP ;        DECIMAL
( Compile to RAM)  1303 LOAD    ( Assembler )   50 LOAD
: ALLOT ( n)    'R +! ;
```

1303 LIST

```
     ( Compile to RAM Changed for 32k appl DW UAH 7-10-89)
 CREATE RAM  40960 ALLOT
: DICTIONARY ( n)    RAM 40960 255 FILL  HDS 2DUP W!  2+ W! ;
: >T ( a - a)    WO @ - 40958 MIN  RAM + ;

: TC@ ( a - n)    >T C@ ;           : TC! ( n a)    >T C! ;
: TW@ ( a - n)    >T W@ ;             : TW! ( n a)    >T W! ;
: TW+! ( n a)    >T W+! ;             : T! ( d a)    >T ! ;
: TC,    HERE TC!  1 'H +! ;    : W,    2 GAP  HERE 2- TW! ;
: ,    4 GAP  HERE 4- T! ;
: CMOVE ( s d n)    >R  >T R> CMOVE ;
: TDUMP ( a n)    OVER + SWAP DO  CR I 7 U.R  I 20 + 1' MIN  I DO
   I 7 AND  0= 2* SPACES  I TW@ 6 U.R 2 /LOOP  20 +LOOP SPACE ;
: FLUSH   RAM  NEW 20 + NEW DO  DUP I BLOCK 1024 MOVE
   UPDATE  1024 +  LOOP DROP  FLUSH ;
1217 LOAD  ( DATA I/O PROM PROGRAMMER)
```

1304 LIST

```
HELP             Displays these target COMPILER instructions

ENP30 LOAD  Compiles polyFORTH for this environment
New programs are compiled to blocks 584-600

s n TDUMP       Dumps target address space,  n bytes at  s
HERE U.         Current top of target dictionary
THERE U.        Next available target RAM location







       Nucleus  Edit    Ass        Total
```


ORIGINAL PAGE IS
OF POOR QUALITY

```
1305 LIST

( ENP-30 2nd load screen loaded from 1306)
CODE X    HERE 10 - HEAD !    HEAD DUP 4+ 28 MOVE
     HEX 8020   HERE 6 - TW!    DECIMAL
( Nucleus)   63 69 THRU    108 109 THRU    70 71 THRU
( Target compiler)   51 53 THRU        FORGET VARIABLE
( Level 1)   72 74 THRU   ( Multiprogrammer)   1275 1277 THRU
   HEX  HERE .   THERE .   DECIMAL
 ( CMOVE)   58 LOAD 1010 WIDTH W!
 2000 LOAD
  2129 LOAD
```

```
1306 LIST

( ENP-30 Load Block 6/1/85)
HELP CR     HOST DEFINITIONS    FORGET ALLOT
: ALLOT ( n)     'R +! ;   HEX   F04000 EQU PROM
26 DICTIONARY   PROM WINDOW   PROM 8 + ORG
F04000 EQU FIRST DECIMAL
  FIRST 256 256 + -   DUP EQU 'OPERATOR    256 +    'R !
1305 LOAD
( Terminal) ( 81 83 THRU   79 LOAD)
( Disk) ( 96 98 THRU)   ( Init Terminal)  ( 80 LOAD)
     1307 LOAD          1308 LOAD
( HEX 1000 H'S T!   1000 H'S 4 + T! DECIMAL)
 HERE H'S T!   HERE H'S 4 + T!
( Links)  HOST   HEAD ram 2+ 32 CMOVE
   HEX  HERE .   THERE .    DECIMAL
  HOST     FLUSH
FORTH    CR CR CR    TODAY W@ .DATE   2 SPACES   @TIME .TIME
```

```
1307 LIST

( Initial RAM values screen loaded from screen 1306) HEX
CREATE OPERATOR     'OPERATOR ,     'OPERATOR BO - ,
    4EF9 W,   'OPERATOR ,   0 , 0 W, ( mmu) 0 ,     'OPERATOR BO - .
    0 W,   0 ,   40 W,    0 ,    0 ,
     0 ,    0 ,    0 ,   FFFC13 ( FF7052) ,
    0 ,    LABEL H'S 8 GAP   0 , 0A W,
 CREATE ram      1 W, 20 GAP   1 W,
       0 ,   0 ,   0 ,   0 ,
       0 ,   0 ,   0 ,   0 ,   0 W,
```


ORIGINAL PAGE IS
OF POOR QUALITY

1308 LIST

```
( ENP30 power up screen loaded from 2028) HEX

LABEL START 3   ram GOLDEN 48 MOVE
   OPERATOR 8 + STATUS  DUP 100 ERASE  46 CMOVE
   set enp   2a_mem  buf_init     do rf[illegible]
   go_lance RUN INIT
   BEGIN     xmit st chknew [illegible]          [illegible]
    clr st  chknew cnkold
                [illegible] @ 1+ <alive !
        AGAIN ;
LABEL [illegible]
(    OA[illegible] # R MOV  2000 #W MTSR)
     [illegible] # 1 MOV
   OPERATOR DUP # U MOV   80 - # S MOV   U R MOV
    NEXT # A2 MOV   D2 S -) MOV   NEXT
 HERE   PROM ORG   0 ,  POWER-UP ,    ORG
```

1309 LIST

```
( Move code to ENP-30)  HEX

: w@enp     F05000 + TW@ ;
: w!enp     C5000 + W! ;
: enpmov  DO I 2 *  w@enp  I 2 *  w!enp  LOOP  ;
: goenp  C5004 W@ C1004 W!  C5006 W@ C1006 W!  8080 C1000 W! ;
```

1310 LIST

ORIGINAL PAGE IS
OF POOR QUALITY

LS11 LIST

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

LS12 LIST

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

LS13 LIST

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

ORIGINAL PAGE IS OF POOR QUALITY

1314 LIST

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

1315 LIST

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

1316 LIST

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

1317 LIST

1318 LIST

1319 LIST

ORIGINAL PAGE IS OF POOR QUALITY

1998 LIST

1999 LIST

2000 LIST

```
( ENP30 INITIALIZATION CONTROL BLOCK CONSTANTS) HEX
8000 CONSTANT lancmode     F01004 CONSTANT inicb
0010 CONSTANT RCBC            0010 CONSTANT TCBC
( F01100 CONSTANT rxhndlr)        ( F01180 CONSTANT txhndlr)
           F01300 CONSTANT ini_b
FFF001 CONSTANT MPUcsr               1E CONSTANT MPUset
F01322 CONSTANT rsp_b
  F01114 CONSTANT x_done

F01120 CONSTANT r_bcb     F01160 CONSTANT head_r
F01164 CONSTANT tail_r
  DECIMAL  2001 2006 THRU
```


ORIGINAL PAGE IS OF POOR QUALITY

2001 LIST

```
( ENP30 Kernel call routine in assember May 14 89) HEX
CODE call_enp   ( parm@ sub@ - parm@ D0)
        S )+ A0 MOV    S )+ R -) MOV
         A0 ) JSR   R )+ S -) MOV   D0 S -) MOV   NEXT
```

2002 LIST

```
( ENP-30 Ethernet board appl. blocks  3  Jun-06-85) HEX
HERE ASSEMBLER
  F01164 AB A0 MOV    ( get bcb tail )
   4 R)   A0 )   MOV    ( move new rcv bcb to tail )
  4 #0   A0       ADD     ( bump tail pointer )
  F01160 # A0 CMP 0= IF   ( chk if wrap around)
    F01120 #   A0    MOV   ( wrap around to begin of fifo )
  THEN
F01160   AB A0   CMP   ( check if head = tail )
0= IF RTS   THEN   ( head = tail = overrun,don't update ptr. )
A0  F01164 AB MOV RTS   ( update tail pointer )
CONSTANT rxhndlr
```

2003 LIST

```
( ENP-30 Ethernet board appl.   mcccccc  3  Jun-06-85) HEX
HERE ASSEMBLER
      1 # F01014 AB MOV   RTS
CONSTANT txhndlr
```

2004 LIST

```
( ENP-30 Ethernet board appl.  mcccccc  3  Jun-06-85) HEX

HERE ASSEMBLER
       RTS
CONSTANT bushndlr
```

2005 LIST

```
( ENP30 ICB setup for UAH amps protocol April 5 89) HEX
: enpmask ( adr --MultiBus_@_space )
    OFFFF AND A0000 + ; ( mask   to enp MultBus adr space )
: ENPmask  ( adr -- ENP_@_space )
    OFFFF AND F00000 + ;  ( mask to ENP30 private adr space)
: MSBC1mask  ( adr -- MultiBus_@_space )
    OFFFF AND 20000 + ; ( mask   to MSBC1 MultBus adr space )

A04B0 CONSTANT SUBAD
: set_enp MPUset MPUcsr C!  ini_b 34 ERASE
ini_b lancmode OVER W! RCBC OVER 2 + W! TCBC OVER 4 + W!
      rxhndlr OVER 10 + ! txhndlr OVER 14 + !
 bushndlr SWAP 18 + ! ini_b F01004 @ DUP SUBAD ! call_enp
        SWAP 1C + 18 CMOVE   ;

: ini_fifo r_bcb 40 ERASE r_bcb DUP head_r ! tail_r ! ;
```

2006 LIST

```
 DECIMAL

 2025 LOAD    2028 LOAD  2030 LOAD
```


ORIGINAL PAGE IS
OF POOR QUALITY

2022 LIST

2023 LIST

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

2024 LIST

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

OF POOR QUALITY

2025 LIST

```
( Kernel response block address constants and variables 6-11-89)
( HEX
F01340 CONSTANT <intercom          F01350 CONSTANT <alive
F01354 CONSTANT <krnlstat             F01358 CONSTANT <netstat
VARIABLE fcode                     VARIABLE csreg)
```

2026 LIST

```
EXIT
```

2027 LIST

```
EXIT
```


ORIGINAL PAGE IS
OF POOR QUALITY

2049 LIST

2050 LIST

```
( DEFINE CONSTANT 1 ---- SL 5-24-89) HEX

 04 CONSTANT ASCN   16 CONSTANT HLCN
 0A CONSTANT AKCN  0B CONSTANT CCN   0C CONSTANT NICN
 0D CONSTANT NIACN   0E  CONSTANT NICL
 04 CONSTANT RTFOS  06 CONSTANT TOFOS
 08 CONSTANT TBAFOS    06 CONSTANT TBLFOS   4 CONSTANT TBSFOS
 00 CONSTANT ANAFOS   06 CONSTANT ANNFOS
 07 CONSTANT ANTPNFOS    08 CONSTANT ANRPNFOS
 00 CONSTANT DAFOS   06 CONSTANT SAFOS    0C CONSTANT AKFOS
 0D CONSTANT SNFOS   0E CONSTANT PNFOS    0F CONSTANT LDFOS
   16 CONSTANT DLFOS     0A CONSTANT ANLCN
08 CONSTANT RBAFOS  06 CONSTANT RBLFOS  04 CONSTANT RBSFOS
 1 CONSTANT TRUE      0 CONSTANT FALSE
4000 CONSTANT TCVERRCN    4000 CONSTANT RCVERRCN
```

2051 LIST

```
( DEFINE VARIABLE) HEX
VARIABLE SNCN
VARIABLE ITCBSPP  VARIABLE ITCBSHD  VARIABLE ITCBSTL
VARIABLE IBSPP  VARIABLE IBSHD  VARIABLE IBSTL
VARIABLE ANASPP  VARIABLE ANASHD  VARIABLE ANASTL
VARIABLE CBSIN  VARIABLE CBSOT  VARIABLE CBSHD  VARIABLE CBSTL
( VARIABLE RCBSIN  VARIABLE RCBSOT  VARIABLE RCBSHD
VARIABLE RCBSTL)  VARIABLE WFASIN  VARIABLE WFASOT
VARIABLE WFASHD   VARIABLE WFASTL
 VARIABLE SACNL  VARIABLE SACNL2    VARIABLE SACNH
  VARIABLE  FLAG
  VARIABLE TOBUF
  VARIABLE tp
VARIABLE RCBSIN  VARIABLE RCBSOT  VARIABLE RCBSHD
 VARIABLE RCBSTL
```

2052 LIST

```
( DEFINE MAILBOX OFFSET  )   HEX

     A0230   CONSTANT MAILBOX
  0 CONSTANT SBFGOS
  4 CONSTANT RDFGOS
  8 CONSTANT SBADOS
  0C CONSTANT MBSNCNOS
  10 CONSTANT KaliveOS
  14 CONSTANT ERROS
  18 CONSTANT NEWBUF
  1C CONSTANT OLDBUF  2C CONSTANT FLOS
   20 CONSTANT oldtmp   24 CONSTANT sah    28 CONSTANT sal
  30 CONSTANT sal2
A0530 CONSTANT mbx   A04C8 CONSTANT rtmp
  A0430 CONSTANT rvbox  A0490 CONSTANT trans
 A0480 CONSTANT STSAD  A04A0 CONSTANT recrv
```

2053 LIST

```
EXIT
```

2054 LIST

```
( PUSH AND POP OPERATION)


: DOPUSH    ( addr SP --   )
         DUP @ ASCN + DUP ROT ! ! ;
: PUSH      ( addr NO SP TL ---  )
         @ OVER @ = IF   DROP  MAILBOX ERROS + !
         ELSE SWAP DROP DOPUSH THEN ;
: DOPOP      ( SP --- addr  )
         DUP @ @ SWAP DUP @ ASCN - SWAP ! ;
: POP        ( NO SP HD --- addr  )
       @ OVER @ SWAP < IF  DROP MAILBOX ERROS + !
         ELSE SWAP DROP DOPOP THEN ;
```

2055 LIST

```
( PUSH_POP OPERATION ON STACKS)


: PUSH_ITCBSK      ( addr ---  )
               1 ITCBSPP ITCBSTL PUSH ;
: POP_ITCBSK   (  --- addr )    2 ITCBSPP ITCBSHD POP ;

: PUSH_IBSK        ( addr ---  )
         3 IBSPP IBSTL       PUSH       ;
: POP_IBSK   (    ---  addr )
         4 IBSPP IBSHD       POP        ;
```

2056 LIST

```
( PUSH_ANASK PROCEDURE)

: PUSH_ANASK      ( addr ptr  ---   )
        >R    DUP SAFOS + @ I ANAFOS + !
        DUP   SAFOS ASCN + + W@ I ANAFOS ASCN + + W!
        DUP   SNFOS + C@ I ANNFOS + C!
        PNFOS + C@ 1 ANRPNFOS + C!
          0    R>  ANTPNFOS + C!    ;

: TEST_ANASK       (     ---     )
          ANASPP DUP @ ANLCN + DUP ROT ! ANASTL @ >
          IF      5  MAILBOX ERROS + !   THEN ;
```

2057 LIST

```
( QUEUE OPERATION 1, QUEUE_IN - JOIN QUEUE)


: ?QUEUE_FULL   ( IN OUT -- f )
              @ SWAP @ ASCN + - 0= ;

: ENTER_QUEUE   ( addr HD TL  IN  ---  )
     DUP DUP @ ASCN + SWAP ! >R ROT I @  ! R>  DUP @ ROT @ =
         IF SWAP @ ASCN - SWAP ! ELSE DROP DROP THEN ;

: QUEUE_IN    ( addr HD TL IN NUMB IN OUT ---   )
       ?QUEUE_FULL  IF  MAILBOX ERROS + !
             ELSE DROP ENTER_QUEUE THEN ;
```

2058 LIST

```
( QUEUE OPERATION 2, QUEUE_OUT - SERVE QUEUE)

: ?QUEUE_EMPTY   ( IN OUT -- f ) @ SWAP @ - 0=    ;
: DELETE_QUEUE   ( HD TL  OUT  -- addr  )    DUP DUP DUP
     @ ASCN + SWAP ! @ @ >R  DUP @ ROT @ =
     IF SWAP @ ASCN - SWAP ! ELSE DROP DROP THEN R> ;
: QUEUE_OUT        ( HD TL OUT NUMB IN OUT ---  addr   )
      ?QUEUE_EMPTY IF  MAILBOX ERROS + !
       ELSE DROP  DELETE_QUEUE  THEN ;
```

2059 LIST

```
( JOIN & SERVE OPERATION ON QUEUE)
HEX
: QUEUE_IN_CBSK      ( addr --  )
     CBSHD CBSTL CBSIN 6 CBSIN CBSOT   QUEUE_IN ;
: QUEUE_OUT_CBSK      (  -- addr  )
    CBSHD CBSTL CBSOT 7 CBSIN CBSOT    QUEUE_OUT ;
: QUEUE_IN_RCBSK      ( addr --   )
    RCBSHD RCBSTL RCBSIN 8 RCBSIN RCBSOT   QUEUE_IN ;
: QUEUE_OUT_RCBSK      (  --- addr  )
    RCBSHD RCBSTL  RCBSOT 9 RCBSIN RCBSOT   QUEUE_OUT ;
: QUEUE_IN_WFASK      ( addr  --   )
    WFASHD WFASTL WFASIN A WFASIN WFASOT   QUEUE_IN ;
: QUEUE_OUT_WFASK      (  --- addr  )
     WFASHD WFASTL WFASOT B WFASIN WFASOT    QUEUE_OUT ;
```

2060 LIST

```
( WORD DEFINITIONS --- COMMON DEFINE WORD   SL  6-01-89)
HEX

: >B_BACK   ( addTC addIB  ---   )     PUSH_IBSK   PUSH_ITCBSK ;

: >ICVR        (  -- addTC  addIB )
               POP_ITCBSK
               POP_IBSK  OVER   TBAFOS + OVER SWAP !
               OVER  HLCN  SWAP TBLFOS + W!
                ;

: !SACN        F0131C W@ SACNH W!   F0131E  @ SACNL !   ;
: !SNCN       0 MAILBOX  MBSNCNOS + ! ;
```

2061 LIST

```
( WORD DEFINITIONS ---- COMMON DEFINE WORD   SL  6-01-89)


: FILL_PACKET    ( PN AK DAH DAL addr --   )
         >R        I  DAFOS + 2 +  !
                   I  DAFOS   + W!
                   SACNL @  I  SAFOS + 2 + !
                   SACNH W@  I  SAFOS  + W!
                   I  AKFOS + C!
                   I  PNFOS + C!
                   SNCN @  R>  SNFOS + C!
       ;
```

2062 LIST

```
( SET UP MAIL BOX )

: GETNAME    (   ---   )
        MAILBOX  MBSNCNOS + @ SNCN ! ;

: MAILBOX_SET_UP      (    ---      )
      MAILBOX  >R
         0 I NEWBUF + !   0 I OLDBUF + !
         0  R>  ERROS + ! ;
```

2063 LIST

```
( KERNEL CALL RECEIVE, TRANSMIT, TIMEOUT SUBROUTINE --- SL  )
HEX

: KEL_RECV  ( addr --     )
      rsp_b  6  + @     call enp DROP DROP ;

: KEL_XMIT    ( addr --     )
      rsp_b 0A  +  @    call_enp   DROP QUEUE_IN_WFASK
  ;
: KEL_TIMEOUT  ( addr   --     )
      rsp_b  0E  +  @  call_enp  DROP DROP ;
```

2064 LIST

```
( OPERATION OF IBSk, NEW, OLD BUFFER)   HEX

: chknew    MAILBOX NEWBUF + @ NOT  IF POP_IBSK
        OFFFF AND C0000 +  MAILBOX NEWBUF + ! THEN ;
: chkold     MAILBOX OLDBUF + @ IF MAILBOX OLDBUF + @ PUSH_IBSK
             0 MAILBOX OLDBUF + ! THEN ;
```

2065 LIST

```
( FREE MEMORY SPACE ---  PLACE A NULL WORD   SL 6-05-89)


: @FREE ;
```

2066 LIST

```
( DEFINE CONSTANT  2) HEX

F08004 CONSTANT BASEADD   0 CONSTANT EPSC
  10 CONSTANT ANACN  20 CONSTANT BCN
100 CONSTANT BLCN  05 CONSTANT CBSCN
14 CONSTANT ITCBSCN   16 CONSTANT IRBLCN   22 CONSTANT ITCBLCN
40 CONSTANT XMCN    10 CONSTANT TCBCN
10 CONSTANT TOCBLCN
14 CONSTANT WFASCN     08 CONSTANT TOSAFOS
0C CONSTANT RMLFOS
22 CONSTANT ICBLCN     10 CONSTANT RCBCN    14 CONSTANT RCBSCN
10 CONSTANT RCBLCN    10 CONSTANT TCBLCN    08 CONSTANT SCBLCN
20 CONSTANT ICBLCN     38 CONSTANT MAILCN
0 CONSTANT TOSET     2 CONSTANT TOCN
```

2067 LIST

```
( DEFINE VARIABLE  2        SL  6-05-89)

 VARIABLE BUFVR
   VARIABLE t_mem
      VARIABLE SCBUF  VARIABLE ICBUF
  VARIABLE  i
```

2068 LIST

```
( MAKE STACKS, ITCBSK IBSK ANASK STACKS) HEX

: MKSTACKS      (  --  )
     BASEADD
     DUP ITCBSHD !  DUP ASCN - ITCBSPP !
     ITCBSCN ASCN * + DUP  ASCN - ITCBSTL !
     DUP IBSHD !   DUP ASCN - IBSPP !
     BCN ASCN * + DUP  ASCN - IBSTL !
     DUP ANASHD !   DUP ANLCN - ANASPP !
     ANACN ANLCN * + DUP  ANLCN - ANASTL !
     BUFVR ! ;
```

2069 LIST

```
( MAKE QUEUES,  CBSK RCBSK WFASK QUEUES) HEX

: MKQUEUES      (  --  )
    BUFVR @ DUP CBSHD !   DUP CBSIN !  DUP CBSOT !
    CBSCN ASCN * + DUP  ASCN - CBSTL !
    DUP RCBSHD !  DUP RCBSIN !   DUP RCBSOT !
    RCBSCN ASCN * + DUP  ASCN - RCBSTL !
    DUP WFASHD !  DUP WFASIN !  DUP WFASOT !
    WFASCN ASCN * + DUP  ASCN - WFASTL !
    BUFVR ! ;
```

2070 LIST

```
( MAKE BUFFERS - RCBK, TCBK, IBSK BUFFER)


: MKBK     ( addr -- addr   )
      RCBCN 0 DO   DUP QUEUE_IN_RCBSK RCBLCN +   LOOP
      TCBCN 0 DO   DUP PUSH_ITCBSK TCBLCN +   LOOP
      BCN 0 DO    DUP PUSH_IBSK   BLCN + LOOP ;
: TISIBUF       ( addr   --   addr  )
      DUP TOBUF !   TOCBLCN +
      DUP SCBUF !   SCBLCN +
      DUP ICBUF !   ICBLCN + ;

: MKBUFFERS      (   --   )    BUFVR @ MKBK   TISIBUF   BUFVR ! ;
```

2071 LIST

```
(  initialize buffers)

: buf_init    MKSTACKS MKQUEUES MKBUFFERS ;
```

2072 LIST

```
EXIT
```

2073 LIST

```
EXIT
```

2074 LIST

```
( 1.6 CALL KERNEL RECEIVE RING FILL --- SL 5-30-89)
HEX
: RING_FILL      ( TEMP --   TEMP f )
        DUP  RCBSOT @  =  IF TRUE
        ELSE     QUEUE_OUT_RCBSK     DUP
                 BLCN SWAP  RBLFOS +  W!
                 DUP POP_IBSK SWAP RBAFOS + !
              rsp_b 06 + @ call_enp DROP DROP
                   FALSE   THEN ;
: do_rfill    (  ---  )
          RCBSIN @
          BEGIN     RING_FILL      TRUE =    UNTIL  DROP
          ;
```

2075 LIST

```
( 1.7 NETWORK INITIALIZATION PROCEDURE 1 --- SL 5-31-89)
 HEX

: BCAST INIT      ( addr --    )
               >R
                 0 NICN  FFFF FFFFFFFF
               R>
               FILL_PACKET ;
```

2076 LIST

```
( 1.7 NETWORK INITIALIZATION PROCEDURE 2 --- SL 5-31-89)
HEX

: ?STAT_ERR      ( addTC --   addTC )
        rsp_b OA + @ call_enp DROP QUEUE_IN_WFASK
          BEGIN    WFASIN WFASOT ?QUEUE_EMPTY   NOT UNTIL
          QUEUE_OUT_WFASK   DUP   TBSFOS + W@
          TCVERRCN AND O = FLAG ! FLAG @ MAILBOX FLOS + !
          ;
```

2077 LIST

```
EXIT
```

2078 LIST

```
( INITIALIZATION --- 1./ NETWORK INITIALIZE    SL 6-02-89)

 HEX
: NETINIT        (    ---     )
          >TCVR   (    --- TCaddr IBaddr)
          BCAST_INIT
         DUP TBSFOS + 2300 SWAP W!
         DUP OC + O SWAP W!
          XMCN O DO
 DUP STSAD !   ( testing program)
                 ?STAT_ERR  FLAG @   IF LEAVE THEN
                 LOOP
          FLAG @ IF QUEUE_IN_WFASK
           ELSE  C MAILBOX ERROS + !        THEN ;
```

2079 LIST

```
( INITIALIZATION --- 1.8 XMITTER TIMEOUT    SL 6-02-89)

: !XM_TIMEOUT      (    ---    )
            TOBUF @ DUP RTFOS + TOSET SWAP W!
            DUP  TOFOS + TOCN SWAP W!
            KEL_TIMEOUT  ;
```

2080 LIST

```
( INITIALIZATION ---  FILL 0 TO  MEMORY LOCATION  SL 6-09-89)
 HEX

: I_MEM
          ITCBSCN ASCN * BCN ASCN * +
          ANACN ANLCN * + CBSCN ASCN * +
          RCBSCN ASCN * + WFASCN ASCN * +
          RCBCN RCBLCN * + TCBCN TCBLCN * +
          BCN BLCN * + 14 ASCN * + 14 ASCN * + ASCN +
           t_mem  ! ;
: >i_mem    I_MEM     BASEADD t_mem @   0  FILL ;
```

2081 LIST

```
EXIT
```


ORIGINAL PAGE IS OF POOR QUALITY

2082 LIST

```
EXIT
```

2083 LIST

```
( TESTING --- INITIALIZATION STATE     SL 6-23-89)

: RUN_INIT
   0 tp !
   0 rtmp !
    0 rvbox !
       !SACN !SNCN
    MAILBOX_SET_UP    GETNAME
      MAILBOX MBSNCNOS + @ EPSC = IF NETINIT THEN
    !XM_TIMEOUT
    1 MAILBOX RDFGOS + !
    SACNH W@ MAILBOX sah + W!  SACNL @ MAILBOX sal + ! ;
```

2084 LIST

```
EXIT
```

2085 LIST

0 EXIT
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

2086 LIST

0 EXIT
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

2087 LIST

0 EXIT
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

2088 LIST

```
EXIT
```

2089 LIST

```
EXIT
```

2090 LIST

```
( RECEIVE STATE --- 2.1 RECEIVE ACKNOWLEDGEMENT  SL 6-01-89)
: PACK_QUE_OUT   ( addRB temp   --  addRB temp )
    >R >R QUEUE_OUT_WFASK DUP  TBAFOS + @ DUP LDFOS + C@
         I SNFOS + C@ =   IF DUP  PNFOS + C@ I PNFOS + C@
         -    DUP 0 <  SWAP DUP 0 = ROT OR   SWAP -4 > AND
              IF >B_BACK
              ELSE  DROP  QUEUE_IN_WFASK  THEN
         ELSE DROP  QUEUE_IN_WFASK  THEN   R> R> ;

: ACK_RECV     ( addRB    --     )
       WFASIN @
       BEGIN
             DUP WFASOT @    =  IF TRUE
             ELSE  PACK_QUE_OUT FALSE  THEN
       UNTIL  DROP  PUSH_IBSK ;
```

ORIGINAL PAGE IS
OF POOR QUALITY

2091 LIST

```
EXIT
```

2092 LIST

```
( RECEIVE STATE ---- 2.2 RECEIVE COMMAND  SL 6-01-89)

: GENERATE_ACK    ( addRB  --     )
    >TCVR    (     ---   addTC   addIB )    >R
          SWAP DUP  PNFOS + C@ AKCN
          ROT  SAFOS + DUP ASCN + W@  SWAP @
    R>    FILL_PACKET    KEL_XMIT     ;

: CHK_COMMAND     ( addRB pointer      --     )
      ANRFNFOS + DUP C@ ROT  DUP PNFOS + C@ ROT SWAP OVER =
      IF     1 + ROT C!  DUP QUEUE_IN_CBSK GENERATE_ACK
      ELSE OVER  PUSH_IBSK OVER PNFOS + C@ SWAP - DUP 0 < SWAP
      -4 > AND IF  GENERATE_ACK ELSE DROP THEN DROP THEN ;
```

2093 LIST

```
( RECEIVE STATE ---- 2.2 RECEIVE COMMAND  SL 6-01-89)


: >SEARCH_CCN   ( addRB    ---      )
      DUP  SNFOS + C@   >R
      ANASHD @
      BEGIN
           DUP ANASPP @ >      IF
                DROP  PUSH_IBSK TRUE
           ELSE   DUP   ANNFOS + C@ I =      IF
                CHK_COMMAND   TRUE
           ELSE   ANLCN +   FALSE   THEN THEN
      UNTIL  R> DROP ;
```


ORIGINAL PAGE IS
OF POOR QUALITY

2094 LIST

```
( RECEIVE STATE --- 2.2 RECEIVE COMMAND  SL 6-01-89)

: CCN_RECV       ( addRB  --      )
          CBSIN CBSOT ?QUEUE_FULL     IF   PUSH_IBSK
          ELSE
               >SEARCH_CCN
          THEN ;
```

2095 LIST

```
EXIT
```

2096 LIST

```
EXIT
```

2097 LIST

```
EXIT
```

2098 LIST

```
( RECEIVE STATE --- 2.3 RECEIVE NICN,  NIACN  SL 6-01-89)


: ?ANASK_FOUND   ( addRB   --   addRB   )
       DUP DUP SNFOS + C@  >R  ANASHD @
       BEGIN
          DUP ANASPP @ >    IF    TEST_ANASK TRUE
          ELSE   DUP   ANNFOS + C@ I =   IF TRUE
          ELSE  ANLCN +  FALSE   THEN  THEN
       UNTIL R>  DROP   PUSH_ANASK  ;
```

2099 LIST

```
( RECEIVE STATE --- 2.3 RECEIVE NICN   SL 6-01-89)
HEX

: ?IS NICN    ( addRB   --    addRB )
        >TCVR    (    --   addTC  addIB )
        >R  OVER SNFOS + C@ I LDFOS +  C!
            OVER 0 NIACN ROT SAFOS + DUP ASCN + W@ SWAP @
        R>   FILL_PACKET
      rsp_b 0A  +  @    call_enp   DROP QUEUE_IN_WFASK
   ;
```

2100 LIST

```
( RECEIVE STATE --- 2.3 RECEIVE NICN    SL 6-01-89)


: NICN_CHK    ( addRB  ---      )
              ?ANASK_FOUND
              ?IS NICN
               PUSH_IBSK
       ;
```

2101 LIST

```
( RECEIVE STATE --- 2.3 RECEIVE   NIACN   SL 6-01-89)


: NIACN_CHK    ( addRB  ---      )
              DUP    GENERATE_ACK
              ?ANASK_FOUND
               PUSH_IBSK
   ;
```

2102 LIST

```
( RECEIVE STATE --- 2.3 RECEIVE  NICL    SL 6-01-89)


: 1CLEAR_WFASK    (      ---       )
          WFASIN @
          BEGIN
             DUP    WFASOT @ = DUP IF
             ELSE   QUEUE_OUT_WFASK   DUP   TBAFOS + @
             >B_BACK   THEN
          UNTIL DROP ;
```

2103 LIST

```
( RECEIVE STATE --- 2.3 RECEIVE  NICL   SL 6-01-89)
: 2CLEAR_RCBSK    (        --        )
        RCBSIN @
        BEGIN
              DUP  RCBSOT @  = DUP IF
ELSE
   RCBSHD RCBSTL  RCBSOT 21 RCBSIN RCBSOT   QUEUE_OUT
   rsp_b  6  + @ call_enp  DROP DROP
                    THEN
        UNTIL DROP ;
```

2104 LIST

```
( RECEIVE STATE --- 2.3 RECEIVE  NICL   SL 6-01-89)
: 3CLEAR_CBSK    (        --        )
        CBSIN @
        BEGIN
              DUP  CBSOT @  = DUP IF
              ELSE  QUEUE_OUT_CBSK   PUSH_IBSK  THEN
        UNTIL DROP ;
```

2105 LIST

```
( RECEIVE STATE --- 2.3 RECEIVE NICL    SL 6-01-89)
: CLEAR_PROTOCOL    ( addRB  --        )
              DUP  GENERATE_ACK
               1CLEAR_WFASK
               ANASHD @ ANLCN - ANASPP !
               2CLEAR_RCBSK
               3CLEAR_CBSK
               PUSH_IBSK
          ;
```

2106 LIST

```
EXIT
```

2107 LIST

```
( RECEIVE STATE  OPERATION 1 ---      SL 6-01-89)


: COMMAND_0_OUT     (    ---   addRB   AKFd    STAT )
    RCBSHD RCBSTL  RCBSOT 25 RCBSIN RCBSOT    QUEUE_OUT
         DUP  RBAFOS + DUP
       @  SWAP   POP_IBSK SWAP !  SWAP DUP
       RBSFOS + W@ SWAP
         rsp_b 6 + @ call_enp DROP DROP
       OVER AKFOS + C@     SWAP  ;
```

2108 LIST

```
( RECEIVE STATE OPERATION 2   ---    SL 6-01-89)


: ?AKFd_CHK      ( addRB  AKFd    --     )
          DUP    AKCN = IF DROP ACK_RECV
          ELSE   DUP   CCN = IF DROP  CCN_RECV
          ELSE   DUP   NICN = IF DROP NICN_CHK
          ELSE   DUP   NICL  = IF DROP CLEAR_PROTOCOL
          ELSE     NIACN = IF      NIACN_CHK
          ELSE    PUSH_IBSK
          THEN THEN THEN THEN THEN ;
```

2109 LIST

```
( receive massage)  HEX
A0478 CONSTANT  tp_tail
: inc_r_head    (    --    )
   head_r @ 4 + head_r - 0=
   IF r_bcb ELSE head_r @ 4 +
   THEN head_r !  ;

: msg_com    (    --    )
  tail_r @  tp_tail !    head_r
     BEGIN
     DUP @ tp_tail @ - 0= IF 1 ELSE DUP @ @ QUEUE_IN_RCBSK
     inc_r_head 0   THEN
  UNTIL  DROP ;
: dsply_msg  OVER trans !  A3000 recrv ! 20 0 DO trans @ W@
  recrv @ W!  recrv @ 2 + recrv ! trans @ 2 + trans ! LOOP ;
```

2110 LIST

```
( RECEIVE STATE OPERATION   ---   SL 6-01-89)
: recv_st     (    --    )
  msg_com
   RCBSIN @
   BEGIN
     DUP   RCBSOT @    = DUP    IF   ELSE
    RCBSOT @ @ rvbox !   ( test for recv msg)
    COMMAND_Q_OUT  RCVERRCN AND NOT  IF dsply_msg ?AKFd_CHK
     ELSE DROP PUSH_IBSK  THEN
   THEN
   UNTIL  DROP
 ;
```

2111 LIST

```
EXIT
```

```
2112 LIST

 EXIT


2113 LIST

 ( TRANSMIT STATE ---  DEFINE VARIABLE -- FLAGS    SL 5-31-89)

 HEX
  8000 CONSTANT  TODONE


2114 LIST

 ( TRANSMIT STATE ---- SEARCH ANASK 1    SL 5-31-89)

 : PACKET_FOUND        ( addTC   addIB   temp@   --      )
           SWAP   >R   >R     I ANTPNFOS + DUP C@
           DUP 1 + ROT C!    CCN
           R> ANAFOS + DUP ASCN + W@ SWAP @    R>
           FILL_PACKET
           KEL_XMIT         ;
```

2115 LIST

```
( TRANSMIT STATE ---- SEARCH ANASK 2       SL 5-31-89)

: #SEARCH    ( addTC  addIB LDVR temp@   --   f )
     DUP ANASPP @    >  IF  DROP DROP    >B_BACK   TRUE
   ELSE  OVER OVER ANNFOS + C@   =
        IF  SWAP DROP  PACKET_FOUND TRUE
    ELSE    ANLCN +  FALSE THEN THEN  ;

: SEARCH_ANASK    ( addTC   addIB   --     )
           DUP LDFOS + C@ ANASHD @
        BEGIN
           #SEARCH
        UNTIL   ;
```

2116 LIST

```
( TRANSMIT STATE ---- 3.1 MAKE PACKET      SL 5-31-89)
  HEX
: MAKE_PACKET      (     ---     )
          POP_ITCBSK DUP  TBAFOS + MAILBOX  SBADOS + @ DUP ROT !
          OVER TBLFOS + OVER DLFOS + C@ HLCN + SWAP W!
          OVER TBSFOS + 2300 SWAP W!
          OVER OC + 0 SWAP W!
          0 MAILBOX  SBFGOS + !
          DROP KEL_XMIT
    (          SEARCH_ANASK )
        ;
```

2117 LIST

```
( TRANSMIT STATE OPERATION   ---   SL 5-31-89)

HEX
: xmit_st    (    ---     )
      MAILBOX  SBFGOS + @ 1 = IF ITCBSPP @ ITCBSHD @ - C > IF
         0 x_done !
      MAKE_PACKET  MAILBOX SBADOS + @ MAILBOX OLDBUF + !
      MAILBOX SBADOS + @ MAILBOX oldtmp + !
      THEN THEN
       tp @ mbx ! tp @ 1 + tp ! ;
```

2118 LIST

```
( TRANSMIT STATE --- TESTING  6-12-89     LAM)
EXIT
```

2119 LIST

```
EXIT
```

2120 LIST

```
EXIT
```

2121 LIST

```
EXIT
```

2122 LIST

```
( CLEAN UP STATE ---- REMOVE ACK, INIT PACKET    SL 5-31-89)
: RM_ACK_INIT    (    --   addTC   addIB   AKFd )
   QUEUE_OUT_WFASK   DUP    TBAFOS + @ DUP AKFOS + C@
  DUP rtmp !   ;
: LOOP_REMOVE    (    ---     )
      RM_ACK_INIT
  DUP AKCN = SWAP NICN = OR  DUP recrv !  IF    >B_BACK
      ELSE  DROP QUEUE_IN_WFASK THEN ;
```

2123 LIST

```
( CLEAN UP STATE OPERATION  ----    SL 6-06-89)
: cln_st     (    ---     )
          WFASIN @
          BEGIN
             DUP  WFASOT @  = DUP IF
             ELSE   LOOP_REMOVE   THEN
          UNTIL DROP ;
```

2124 LIST

```
EXIT
```

2125 LIST

```
( TIMEOUT STATE --- TIMEOUT INTERNAL LOOP      SL 5-31-89)
HEX
: NEG_AKVR    ( addIB  AKFd  ---      )
        NEGATE SWAP AKFOS + C! ;
: TIMER_LOOP    (     ---      )
        QUEUE_OUT_WFASK    DUP   TBAFOS + @
           DUP   AKFOS + C@     DUP 80 AND 0 >
           IF   NEG_AKVR     KEL_XMIT
           ELSE  NEG_AKVR  QUEUE_IN_WFASK   THEN ;

: <TIMEOUT      (      ---      )
        WFASIN @
        BEGIN
             DUP WFASOT @   = DUP   IF
             ELSE   TIMER_LOOP  THEN
        UNTIL   DROP ;
```

2126 LIST

```
( TIMEOUT STATE OPERATION  ---    SL 5-31-89)


: tout_st     (      ---      )
           TOBUF @ RTFOS + DUP W@   TODONE AND  IF
           0 SWAP W!  <TIMEOUT ELSE DROP   THEN ;
```

2127 LIST

```
EXIT
```

2128 LIST

2129 LIST

```
( LOAD PROTOCOL -- SL 7/18/89 )

   2050 2127 THRU
```

2148 LIST

2149 LIST

```
( DEFINE MAILBOX OFFSET  )    HEX
     20230    CONSTANT MAILBOX
  0 CONSTANT SBFGOS
  4 CONSTANT RDFGOS
  8 CONSTANT SBADOS
  0C CONSTANT MBSNCNOS
  10 CONSTANT <aliveOS
  14 CONSTANT ERROS
  18 CONSTANT NEWBUF   30 CONSTANT sal2
  1C CONSTANT OLDBUF   2C CONSTANT FLOS
   20 CONSTANT oldtmp   24 CONSTANT sah   28 CONSTANT sal
16 CONSTANT DLFOS   20490 CONSTANT trans
 20 CONSTANT BCN    20480 CONSTANT S1SAD
  20530 CONSTANT mbx   20430 CONSTANT rvbox
 VARIABLE tmpbuf   204A0 CONSTANT recrv     204C8 CONSTANT rtmp
  204B0 CONSTANT SUBAD 204C0 CONSTANT STAD
```

2150 LIST

```
( Memory MAP: MSBC1 and ENP-30 IPC area) HEX

( >: Host to ENP commands)

20006 CONSTANT >send            20008 CONSTANT >sndaddr
2000C CONSTANT >reset
24020 CONSTANT >bcb             24060 CONSTANT >head
24064 CONSTANT >tail            24068 CONSTANT x_fbh
2406C CONSTANT x_fbt
8080 CONSTANT Go_cmd

0C CONSTANT AKFOS

  0D CONSTANT SNFOS    0E CONSTANT PNFOS    0F CONSTANT LDFOS
```

2151 LIST

```
( Memory MAP: MSBC1 and ENP-30 IPC area) HEX
( <: ENP to Host commands)
20024 CONSTANT <sdone          20026 CONSTANT <sndaddr
23900 CONSTANT <sts ( status from ENP)
2000E CONSTANT >status      2002A CONSTANT <status
2002C CONSTANT <bcb       20070 CONSTANT <head
20074 CONSTANT <tail      20078 CONSTANT fbh_r
2007C CONSTANT fbt_r

21200 CONSTANT <data ( <ENP)  22500 CONSTANT >data ( >ENP)

28000 CONSTANT 'enp          C4000 CONSTANT enp
( 28000 CONSTANT 'enp  for RAM testing)
( F88000 CONSTANT 'enp for PROM base)
( Changed F86000 to F88000 to adjust starting address for new
16k eproms used for UAH ethernet protocol)
```

2152 LIST

```
( Polling ENP to Host Background task)
EXIT
20 64 150 BACKGROUND <netdata ( data from network)
20 64 150 BACKGROUND t_dummy  ( dummy task)
```

2153 LIST

```
( Polling ENP to Host Background Task) HEX
EXIT
: bldenp  <netdata BUILD
            t_dummy BUILD ;
VARIABLE T1  VARIABLE T2 VARIABLE T3 VARIABLE T4
: ?T1 T1 @ . ; : ?T2 T2 @ . ; : ?T3 T3 @ . ; : ?T4 T4 @ . ;
```

2154 LIST

```
( MSBC1 Ethernet data storage initialization) HEX
: ini_fifo  <bcb 44 ERASE  <bcb DUP <head ! <tail ! ;
: iflags 4 <sdone W! 0 <sndaddr ! 0 >sndaddr ! 0 >send W! ;
: erabf   <data 110 10 * ERASE  >data 110 10 * ERASE ;
: ln_set  110 * 6 + + W! ;
: b@_set  110 * 8 + + ! ;
: rx_ini  erabf
          10 0 DO 100 <data I ln_set LOOP
          10 0 DO 100 >data I ln_set LOOP
          10 0 DO <data I 110 * 10 + + <data I b@_set LOOP
          10 0 DO >data I 110 * 10 + + >data I b@_set LOOP ;
: chn>fb  10 0 DO I 1+ 110 * <data +
                 I 110 * <data + ! LOOP
          <data fbh_r !   <data OF   110 * + fbt_r !
          -1 <data OF   110 * + ! ;
: iMSBC1 ini_fifo rx_ini chn>fb iflags ; EXIT
```

2155 LIST

```
( MSBC1 <> ENP30 COMMUNICATION AREA DEFINITIONS) HEX
( ALSO LOCATION AND DEFINITONS FOR STATUS CONTROL BLOCK)
20000 CONSTANT msb_enp  ( beginning of msbc1 <> enp30
communication area)
20002  CONSTANT <alive
20006 CONSTANT RdFA ( ready add) 20008 CONSTANT SBFA ( send add)
A5A5 CONSTANT RdFg ( ready)       5A5A CONSTANT SBFg ( send)
VARIABLE SBAd ( send buffer add) VARIABLE CBSHd ( c buf head)
VARIABLE CBSTl ( cmd buf tail) VARIABLE CBSIn ( cmd buf stk in)
VARIABLE CBSOt ( cmd buf stk out)
VARIABLE IBSHd ( idle buf head) VARIABLE IBSTl ( idle buf tail)
VARIABLE IBSPp ( idle buf ph/pp) VARIABLE IBSRq ( idle buf req)
VARIABLE IBSLk ( idle buf lock)
```

2156 LIST

```
( MSBC1 starting the enp30 5-10-89 dw) HEX
: delay_goenp 0AFFFF 0 DO LOOP ;
: rstenp ." Init. Ethernet Ctr:1. " CR delay_goenp 1 DF001 C! ;
: badenp C1002 C@ DUP 0= IF ABORT" SELFTEST FAIL " THEN ;
: enpmov  2000 0 DO I 2* 'enp + W@  I 2* enp + W!   LOOP ;
: goenp rstenp delay_goenp badenp enpmov
          'enp 4 + W@ C1004 W! 'enp 6 + W@ C1006 W!
            8080 C1000 W!  ;
: !enpsadr 'enp 4 + W@ C1004 W! 'enp 6 + W@ C1006 W! ;

: w@enp  W@ DUP 0FF AND 100 * SWAP FF00 AND 100 / OR ;
: w!enp SWAP DUP 0FF AND 100 * SWAP FF00 AND 100 / OR SWAP W! ;
: @enp  DUP w@enp 10000 * SWAP 2+ w@enp OR ;
: !enp 2DUP SWAP  100 / 100 / SWAP w!enp 2+ SWAP
        0FFFF AND SWAP w!enp ;
```

2157 LIST

```
( MSBC1 communication words  02-26-86 ) HEX
: dmsg   ; ( display msg to CRT )

: inc_head ( -- ) 0 <head   @ !   ( clear buf @ in fifo )
    <head @ 4 + <head - 0= ( Ck if <head at end of fifo)
    IF <bcb ELSE <head @ 4 +   ( det new <head value )
    THEN  <head ! ;      ( update <head value )

: rtn>fb   ( MSBC1_fb@ -- )
   fbh_r @ 1+ 0=
   IF DUP fbt_r !   -1 SWAP  !
   ELSE DUP fbt_r @ ! ( pt last fb to added buf )
   -1 OVER !   ( set added fb to last, -1 )
   fbt_r !     ( update fbt_r ptr )
   THEN fbh_r @ 1+ 0=
   IF fbt_r @ fbh_r ! THEN ;
```

2158 LIST

```
( Test message generation) HEX

: ?msg  ( -- )
 ( BEGIN)  <head @ <tail @ - 0= NOT ( any msg's? )
        IF dmsg          ( display msg to CRT )
                ( place <head value for rtn>buf )
           inc_head      ( adj <head ptr )
           rtn>fb   ( rtn dsply msg buf to fb pool )
        THEN
 ( <head @ <tail @ - 0= UNTIL) ; ( any more msg's ? )

: .msgs ( n -- ) 0 DO ?msg LOOP ; ( display n msg's)
```

2159 LIST

```
( MSBC1 comm test words 3-6-86 ) HEX

: dbuf     ." IBSK buffer " CR CR
        BCN 0 DO  mbx I 4 * + @ . CR LOOP ;
: dnew    ." newbuf = " MAILBOX NEWBUF + @ . CR
     ." oldbuf = " MAILBOX OLDBUF + @ . CR
      ." oldtmp = " MAILBOX oldtmp + @ . CR
      ." Ethernet addr  high byte = " MAILBOX sah + W@ . CR
     ." LOW BYTE = "  MAILBOX sal + @  . CR ;
: dmx     CR CR  ." run times = " mbx @ . CR ;
: drvmsg   CR ." receive message "  CR
     rvbox @ 0FFFF AND C0000 +  tmpbuf ! tmpbuf  @ .  ;
```

2160 LIST

```
( TRANSMIT message generation ) HEX
: bcaddr   FFFF MAILBOX NEWBUF + @ 0 +  W!
           FFFF MAILBOX NEWBUF + @ 2 + W!
           FFFF MAILBOX NEWBUF + @ 4 + W!     ;

: msg 40 0 DO BBBB I 2* MAILBOX NEWBUF + @ + 18 + w!enp
                  LOOP ;
: tmsg   bcaddr msg  MAILBOX NEWBUF + @ DUP  DLFOS +  80
SWAP C! DUP AKFOS + 0C SWAP C! DUP PNFOS + 0 SWAP C! SNFOS + 0
SWAP C! MAILBOX NEWBUF + @ 0FFFF AND F00000 + MAILBOX SBADOS
 + ! ;
: sbmsg     1 MAILBOX SBFGOS + !   ;
: derr   ." erros = " MAILBOX ERROS + @ . CR ." flag = "
       MAILBOX FLOS + @ . CR  ." status " STSAD @ . CR
   ." trans " trans @ . CR ." recrv " recrv @ . CR
 ." SUB = " SUBAD @ .  CR ." STS = " rtmp @ . ;
```

2161 LIST

```
( Debug test tools ) ( 2-4-86 )   HEX
AA55 CONSTANT Sflg
: dly_sts 01FFFF 0 DO LOOP ;
: .sts ( -- )
     ." >status = " >status W@ . 10 SPACES ." <status = " <status
  W@ . CR <sts 50 HEX DUMP  DECIMAL CR CR  ;

: .alive CR ( -- ) ( show if ENP-30 is running )
." NETWORK PROCESS COUNTER " CR
<alive @ . 30 0 DO LOOP <alive @ . ;

: wmsg 40 0 DO 0 I 2* MAILBOX NEWBUF + @ + 18 + w!enp
                  LOOP ;
: imsg   bcaddr  wmsg 2 MAILBOX NEWBUF + @ AKFOS + C!
        MAILBOX NEWBUF + @ 0FFFF AND F00000 +
        MAILBOX SBADOS + ! ;
```

2162 LIST

```
( Debug test tools ) ( 3-19-86 )
HEX
: smsg   ( a ---- )
    0 DO tmsg 1 MAILBOX SBFGOS +  !  ( SEND MSG )
              LOOP ;

: dp msg     20 0 DO I 2* 23000 + W@ . LOOP ;
```